DE MONTFORT UNIVERSITY

School of Computing Sciences

Department of Mathematical Sciences

Data Compression Schemes for Pattern Recognition in Digital Images using Fractals

Abdul Razag Ali Aburas

Ph.D Thesis

July 1997

Supervisor: Professor J.M. Blackledge

ABSTRACT

Pattern recognition in digital images is a common problem with application in remote sensing, electron microscopy, medical imaging, seismic imaging and astrophysics for example. Although this subject has been researched for over twenty years there is still no general solution which can be compared with the human cognitive system in which a pattern can be recognised subject to arbitrary orientation and scale.

The application of Artificial Neural Networks can in principle provide a very general solution providing suitable training schemes are implemented. However, this approach raises some major issues in practice. First, the CPU time required to train an ANN for a grey level or colour image can be very large especially if the object has a complex structure with no clear geometrical features such as those that arise in remote sensing applications. Secondly, both the core and file space memory required to represent large images and their associated data tasks leads to a number of problems in which the use of virtual memory is paramount.

The primary goal of this research has been to assess methods of image data compression for pattern recoginition using a range of different compression methods. In particular, this research has resulted in the design and implementation of a new algorithm for general pattern recognition based on the use of fractal image compression.

This approach has for the first time allowed the pattern recognition problem to be solved in a way that is invariant of rotation and scale. It allows both ANNs and correlation to be used subject to appropriate pre-and post-processing

techniques for digital image processing on aspect for which a dedicated programmer's work bench has been developed using X-Designer.

Acknowledgements

My grateful thanks to Almighty Allah who gave me the ability, health and knowledge to gain my lifetime achievement of completing my PhD.

I would like to express my gratitude and appreciation to Professor J.M. Blackledge who acted as my Supervisor throughout my research for his assistance, advice, guidance and support. I would also like to thank him for his "magic words" that helped me to solve programming problems which I encountered.

I would like to thank the Head of the Faculty of Engineering at Tajura and also everyone at this establishment for giving me the opportunity to achieve my PhD degree.

I dedicate this manuscript to all my family and friends who stood by me and supported me all the way.

I would also like to express my thanks to Miss Tasneem Mukadam for her help in typing up this thesis.

CONTENTS

Notation

DIP	**Digital Image Processing**
FIR	**Finite Impulse Response**
DFT	**Discrete Fourier Transform**
FFT	**Fast Fourier Transform**
$\hat{F}_1$	**1D Fourier Transform Operator**
$\hat{F}_1^{-1}$	**1D Inverse Fourier Transform Operator**
$\hat{F}_2$	**2D Fourier Transform Operator**
$\hat{F}_2^{-1}$	**2D Inverse Fourier Transform Operator**
s_{ij}	**Recorded Signal/Image**
p_{ij}	**Point Spread Function (PSF)**
f_{ij}	**Object Function**
$\hat{f}_{ij}$	**Estimate of Object Function**
n_{ij}	**Noise Function**
$\otimes\otimes$	**2D Convolution Operation**
$\odot\odot$	**2D Correlation Operation**
S_{ij}	**DFT of s_{ij}**
P_{ij}	**DFT of p_{ij}**
F_{ij}	**DFT of f_{ij}**
$\hat{F}_{ij}$	**DFT of $\hat{f}_{ij}$**
N_{ij}	**DFT of n_{ij}**

NSR	**Noise to Signal Ratio**
SNR	**Signal to Noise Ratio**
λ	**Longrange Multiplier**
E	**Entropy**
PSE	**Power Spectrum Equalization**
BL	**BandLimited**
Sinc	**sinc function [sinc(x)=sin(x)/x]**
δ	**1D Dirac Delta Function**
$\vec{K}$	**2D Spatial Frequency Vector**
HPF	**High Pass Filter**
∇	**Gradient**
∇^2	**Laplacian**
CAT	**Computer Aided Tomography**
O(x, y)	**Object Function**
$\tilde{O}(x, y)$	**DFT of O(x,y)**
$P(Z, \vartheta)$	**Projection of O(x,y)**
$\hat{R}$	**Radon Transform Operator**
$\hat{R}^{-1}$	**Inverse Radon Transform**
$\vec{r}$	**2D Space Vector**
$\hat{\eta}$	**2D Unit Vector**
B(x, y)	**Back Projection of O(x,y)**
$\tilde{B}(x, y)$	**DFT of B(x,y)**
HP	**Hewlett Packard**
X	**X Window System**
OSF	**Open Software Foundation**

GUI	**Graphical User Interface**
API	**Application Programmer's Interface**
∂	**Differential Operator**
BP	**Back Propagation**
$\rightarrow$	**Pathway**

List of Figures

Chapter 1

Basic Concepts

This chapter provides an overview of the project (research). It also gives an introduction about the organisation of the thesis, an idea about each chapter and work bench on which this research has been carried out.

In this chapter **Page No**

1.0. About this Thesis

The original research theme undertaken for this thesis was on the application of digital image compression schemes for training Artificial Neural Networks (ANN) to solve problems associated with pattern recognition. The principal aim has been to study the effect of data compression on the CPU time and memory requirements for feature recognition using ANNs. The justification for this approach was twofold.

(i) Application of ANNs to full grey level images is both time consuming and requires extensive memory resources.

(ii) Data compression schemes are ideally designed to encode image information in a unique and unambiguous way and thus, retain information on those features appropriate to a pattern recognition paradigm.

After designing the backpropagation algorithm for a ANN, a variety of image compression schemes were investigated including the compression transform (JPEG) and fractal compression. In each case, the success or failure of the algorithms developed depends critically on the pre- and/or post-processing techniques applied to a particular image data. Central to this need has been both the development of an image processing package developed using a wide range of algorithms for both low and high level image processing. Brief overviews of these algorithms are presented in this thesis for completeness as is the development of the interface used to drive this software.

This development was accomplished using X-Designer which has provided an ideal environment for undertaking this research, all software having been written from scratch. A variety of useful algorithms have been written during this research and re-changed effectively using X-Designer. The central issue has been the use of image compression schemes for feature recognition which

is invariant of rotation and scale. This problem occurs in a variety of applications but is particularly important in remote sensing where the template may not correlate well with a brown feature because of changes in scale and/or orientation. The approach followed has been to use a fractal compression scheme whereby the fractal parameters are used to trace the ANN and/or correlate with the template after fractal compression.

This approach to feature recognition solves a well known problem which has previously been subject to either excessive CPU times of data error and represents the principal original contribution to the field. This form of 'image processing in compression space' is one of many techniques that could find valuable applications and represents the first of many new paradigms that could be considered for image processing in general. The thesis is presented in a form in which the package developed for this research is merged with other theoretical and technical issues and represents the practical approach to the software engineering that has been executed during the period of research.

1.1. The Organisation of the thesis

This thesis has been divided into eight chapters and seven appendices.

The thesis begins with an introductory chapter which aims to give in brief an idea about the main problem which this research is based on. Also in this chapter, a brief introduction about the contents of each chapter in this thesis .

The aim of this chapter is to give to both the specialist and non-specialist reader some of the basic concepts and provide a structural setting for some of the more specialist material in later chapters. The remaining chapters are divided into sections and each has its own brief introduction.

In chapter 2, I have included the most important of the previous research work papers related to this research, with their authors, their titles, their reference and the dates, so that one included the reader can use it as a reference for further reading. Also included discussion about the research work and what has been achieved .

Chapter 3 summarises the basic mathematical background needed for understanding the algorithms that have been used for building up the DIP Software Package.

Chapter 4 describes the fully connected Artificial Neural Networks that have been implemented for pattern recognition together with the mathematical background supported with diagrams for the ANN model.

In this thesis, Data Image Compression techniques are considered. There are discussed in Chapter 5 and chapter 6. All compression methods are implemented in the DIP Package.

In chapter 5, each compression algorithm is introduced with the appropriate mathematical background followed by a step-by-step analysis algorithm "How it works" supported with some visual results and brief discussion.
The key part of this research (the original contribution in chapter 6) is given which describes the most powerful compression algorithm -Fractal Image Compression - and include mathematical background and step analysis for the algorithm. Iterated Function System (IFS) are also discussed.

In chapter 7, techniques are described that have been used for pattern recognition supported with some results that explain how the fractal compression techniques can be used to solve the rotation/scale invariant pattern recognition problem.

The last chapter gives a conclusion to the thesis and includes a brief overview of the results and recommendations for future research work in both Data Image Compression and Pattern Recognition.

This thesis is supported by seven appendices, the first gives the necessary steps requested to install the DIP Software Package, the second appendix provides an introduction to the X-Designer Software Package that has been used to generate the interface, and work bench on which this research has been carried out, the third explains how to start the application, states the interface features and describes the interface in detail, the forth displays some visual results produced by the DIP functions, the fifth appendix shows all the results of implementing the Automatic Pattern Recognition technique based on fractal image compression algorithm on black and white images and grey scale images. These results have been displayed in 2D space view and 3D space view, the sixth describes the software engineering for the interface flowchart and a brief description about each function that has been used by the DIP Software Package. The final appendix displays the DIP Software Package warning messages that detect any mistakes made by the user.

Chapter 2

Background to Research and Literature

This chapter represents a number of various reference papers dealing with issues that relate to the work in this thesis.

This chapter is divided into four sections as follows:

2.1. Digital Image Processing Papers

In this section, we represents samples of the most important of the outcome from the previous research work papers on digital image processing. These papers are the principal material on which this research thesis has been based.

A Fast Two-Dimensional Median Filtering Algorithm by Thomas S. Hung, George J Yang and Gregory Y. Tang[1].

This paper presented a fast algorithm for two-dimensional median filtering. It is based on storing and updating the grey level histogram of the picture elements in the window. The algorithm is much faster than conventional sorting methods. For a window size of MxN, the computer time required is O(u). Their work has been implemented by the DIP package under Filter option menu.

Advances in Mathematical Models for Image Processing by Anil K. Jain[2].

The paper considered several state-of-the art mathematical models useful in image processing. These models include the traditional fast unitary transforms, autoregressive and state variable models as well as two-dimensional linear prediction models. These models introduced earlier[51], [52] as low-order finite difference approximations of partial differential equations are generalised and extended to higher order in the framework of linear prediction theory. Applications in several image processing problems, includes image restoration, smoothing, enhancement, data compression, spectral estimation, and filter design, are discussed and examples given. Some

of these mathematical models have been implemented by the DIP package under Enhancement option menu.

Digital Image Enhancement and Noise Filtering by Use of Local Statistics by Jong-Sen Lee[3].

His research work, involving contrast enhancement and noise filtering on two-dimensional image arrays are developed based on their local *mean* and *variance*. These algorithms are non-recursive and do not require the use of any kind of transform. They share the same characteristics in that each pixel is processed independently. Consequently, this approach has an obvious advantage when used in real-time digital image processing applications and where a parallel processor can be used. For both the additive and multiplicative cases, the *a priori* mean and variance of each pixel is derived from its local mean and variance. Then, the minimum mean-square error estimator in its simplest form is applied to obtain the noise filtering algorithms. For multiplicative noise a statistical optimal linear approximation is made. Experimental results show that such an assumption yields a very effective filtering algorithm. The techniques developed in this paper are readily adapted and implemented by the DIP package.

2.2. Artificial Neural Networks Papers

In this section, we represents samples of the most important of the outcome from the previous research work papers on Artificial Neural Networks and its applications. This collection is not intended to be an exhaustive listing of all the relative papers but a representative sample.

Comparison of pre-processing Transforms for Neural Network Classification of Character Images by A.Roberts, M.Yearworth[4].

His research work is based on the conjecture that the combination of unitary image transforms, an image processing technique, with neural network classifiers is a potentially useful method for performing OCR. The investigative work consisted of a series of experiments that involved taking live character image data, performing a set of unitary integral transforms on that data, and evaluating the effect of the transforms on the learning behaviour and recognition performance of a fully-connected, three-layer, back propagation network. This was done by comparison with a network trained on the un-preprocessed image data. The results of these experiments indicates that neural networks are capable of learning to recognise character images that have been transformed with similar levels of accuracy to networks trained on un-preprocessed images. It was found that learning rates were somewhat impaired on transformed data, however, and possible explanations are discussed. This particular paper gives the hint of how we can test, train and/or extend our ANN model for pattern recognition.

A Stable Second Order Method for Training Back Propagation Networks by Philip R. Nachtsheim[5].

This paper described a simple method for improving the learning rate of the back-propagation algorithm. The basis of the method is that approximate second order corrections can be incorporated in the output units. The extended method leads to significant improvements in the convergence rate. This paper help to understand the BP algorithm and how it works.

Back Propagation in Perceptrons with Feedback by Luis B. Almeida, R. Alves Redol[6].

This paper presented an extension of back propagation to systems with feedback. The error propagation circuit is interpreted as the transpose of the linearized perceptron network. The error propagation network is shown to always be stable during training, and a sufficient condition for the stability of the perceptron network is derived.

Pattern Recognition by Homomorphic Graph Matching using Hopfield Neural Networks by P N Suganthan, E K Teoh and D P Mital[7].

This paper investigated the application of the Hopfield neural network as a constraint satisfaction network for pattern recognition. Suitable energy and compatibility functions are introduced for pattern recognition by homomorphic attributed relational graph (ARG) matching. In this paper, a neural network initialization strategy is applied to achieve the desired complexity reduction. Further, a method to verify and localize the hypothesis generated by the Hopfield network is also presented using an efficient pose clustering algorithm. The performance of the connectionist approach to pattern recognition by homomorphic relational graph matching is demonstrated using a number of line patterns, silhouette images and circle patterns. The Hopfield neural network is another model widely used for pattern recognition.

Neural Networks for Texture Classification by A K Muhamed and F Deravi[8].

In this paper, they discussed the application of artificial neural networks to texture analysis. Such networks are used as classifiers in order to distinguish among various classes of texture. A set of experiments was carried out in order to study the network's classification performance as a function of both its size and the training volume. Another set of experiments was carried out in order to explore the effect of training strategy on the overall performance. Recognition rates of up to 94.4 percent were achieved for tests made on unseen data. This paper shows different methods for testing the ANN.

A Neural Network Based Position Invariant Pattern Recognition System with a Constrained Hypermedia Style User-Interface by E C Mertzanis, D A Dukic, I D Benest and J Austin[9].

This paper aims to introduce a method capable of recognising an aircraft from any viewpoint in uncluttered noisy scenes. It will also describe a proposed exploitation of a recently developed hypermedia style user-interface for presenting and manipulating information, and for enabling users to specify image processing and pattern recognition operations.

Current neural network based methods are limited by the time required to learn to recognise an object, as well as by the number of pictures required to be processed in order to achieve successful classification results. Typically, a number in the order of thousands of pictures per object can often be required in training multi-layer neural networks. In this paper a new method that uses a single presentation of an image during training has been presented.

The paper divided in two main parts. The first part described the pattern recognition process and included a description of the image processing

procedures, a detailed description of the Quadtree and normalised Quadtree data structures, the theory and properties of the Quadtree data structures, and finally the actual method used in order to train and test the artificial neural network. The second part described an initial presentation of the user-interface graphical environment and its corresponding sub-parts for each individual stage of the pattern recognition process. This particular research work focuses on our research point which is rotation variant problem in pattern recognition and shows why is not an efficient method.

Internal Measuring Models in Trained Neural Networks for Parameter Estimation from Images by Tian-Jin Feng, Z Houkes, M J Korstan and L J Spreeuwers[10].

This paper discusses a possible interpretation of the learned knowledge of a network trained for parameter estimation from images. The output of the hidden layer are the internal components of the output parameters. The input-to-hidden weight maps, functioning as a kind of internal measuring model of the parameter components include statistical features of the training set and seem to have a clear physical and geometrical meaning. This paper gives a clear view of the map of the input-hidden-output layers in the ANN model.

2.3. Image Data Compression Papers

This section represents a number of the most important papers specifically related to Image Data Compression which some of these papers outcome have been implemented in building up the DIP library package.

A Fast Karhunen-Loeve Transform for a Class of Random Processes by Anil K Jain[11].

This paper discussed the Karhunen-Loeve Transform for a class of signals. It is proven to be a set of periodic sinc functions. This Karhunen-Loeve series expansion can be obtained via an FFT algorithm. This fast algorithm obtained could be useful in data compression and other mean-square signal processing applications.

Singular Value Decomposition (SVD) Image Coding by Harry C Andrews, Claude L Patterson[12].

This concise paper presents a new transform method in which the singular values and singular vectors of an image are computed and transmitted instead of transform coefficients. The singular value decomposition (SVD) method is known to be the deterministically optimal transform for energy compaction. A systems implementation is hypothesised and a variety of coding strategies is developed. Statistical properties of the SVD are discussed and a self adaptive set of experimental results is presented. Imagery compressed to 1, 1.5 and 2.5 hits per pixel with less than 1.6, 1 and 1/3 percent, respective mean-square error is displayed. Finally, additional image coding scenarios are postulated for further consideration.

Image Data Compression Using Autoregressive Time Series Models by Edward J Delp, Rangasami L Kashyap and O Robert Mitchell[13].

A two-dimensional image model is formulated using a seasonal autoregressive time series. With appropriate use of initial conditions, the method of least

squares is used to obtain estimates of the model parameters. The model is then used to regenerate the original image. Results obtained indicate this method could be used to code textures for low hit rates or be used in an application of generating compressed background scenes. A differential pulse code modulation (DPCM) scheme is also demonstrated as a means of archival storage of images along with a new quantization technique for DPCM. This quantization technique is compared with standard quantization methods.

Robust Two-Dimensional Tree Encoding of Image by James W Modestino, V Bhaskaran[14].

This paper described a class of robust two-dimensional (2-D) tree coding procedures for efficient encoding of monochrome images. The encoding filter is designed on the basis of ensemble properties of a particular class of stochastic random fields which have been shown to provide a useful model for typical imagery data. They used the (M; K′, L′) algorithm in instrumenting the 2-D tree search. Results are demonstrated for both synthetic images and a number of head-and-shoulders images. These results demonstrate the effectiveness of 2-D tree encoding using fairly moderate search intensities and the inherent robustness of 2-D tree encoding more generally. This particular paper demonstrated the logical design of the encoding filter under an appropriate stochastic modelling assumption, the 2-D tree encoder provides relatively efficient performance for a rather wide range of real-world imagery.

Image Compression Using Block Trunction Coding by Edward J Delp and O Robert Mitchell[15].

This paper presented and compared a new technique for image compression called Block Trunction Coding (BTC) with transform and other techniques. The BTC algorithm uses a two-level (one-bit) non-parametric quantizer that adapts to local properties of the image. The quantizer that shows great promise is one which preserves the local sample moments. This quantizer produces good quality images that appear to be enhanced at data rates of 1.5/picture element. No large data storage is required, and the computation is small. The quantizer is compared with standard (minimum mean-square error and mean-absolute error) one-bit quantizers. Modifications of the basic BTC algorithm are discussed along with the performance of BTC in the presence of channel errors.

The Laplacian Pyramid as Compact Image Code by Peter J Burt, Edward H Adelson[16].

This paper described a technique for image encoding in which local operators of many scales but identical shape serve as the basis functions. The representation differs from established techniques in that the code elements are localized in spatial frequency as well as in space. Pixel-to-pixel correlations are first removed by subtracting a low-pass filtered copy of the image from the image itself. The result is a net data compression since the difference, or error, image has low variance and entropy, and the low-pass filtered image may be represented at reduced sample density. Further data compression is achieved by quantizing the difference image. These steps are then repeated to compress the low-pass image. Iteration of the process at appropriately expanded scales generates a pyramid data structure. The encoding process is equivalent to sampling the image with Laplacian operators of many scales. Thus, the code tends to enhance salient image features. The advantage of the present code is

that it is well suited for many image analysis tasks as well as for image compression.

Digital Video Bandwidth Compression Using Block Trunction Coding by Donald J Healy, O Robert Mitchell[17].

This paper presented a technique for bandwidth compression coding of sequential digitized video imagery. The method uses adaptive one-bit quantization over small blocks of space and time. The system includes global registration and motion detection for optimal bit assignments. A comparison is made with a conditional replenishment technique using a motion compensated predictor described in the literature. The method presented produces reconstructed video sequences filmed from low altitude aircraft with slight degradation which is acceptable for many remote sensing applications at 0.9 hit/pixel data rate. Robust behaviour in the presence of channel noise is demonstrated.

Data Compression of Moving Images Using Geometric Transformations by C A Papadopoulos and T G Clarkson[18].

This paper presented a new compression technique which can be used to considerably reduce the amount of data that are required to digitally transmit or store a sequence of images. The method used to compress the data is based on Image Resampling and describes the new frame as a set of geometric transformations of parts of the previous frame. A modular image processing system is also described which can be used to implement the encoder and decoder for this compression technique.

2.4. Pattern Matching and Pattern Recognition Papers

In this section, we represents samples of the most important of the outcome from the previous research work papers on pattern recognition. These papers are the principal material on which this research work has been based.

Efficient Two-Dimensional Compressed Matching by Amihood Amir and Gary Benson[19].

Digitized images are known to be extremely space consuming. However, regularities in the images can often be exploited to reduce the necessary storage area. In this paper, they found that many systems store images in a compressed form. They proposed that compression could be used as a time saving tool, in addition to its traditional role of space saving. They introduced a new pattern matching paradigm - *Compressed matching*. A text array T and Pattern array P are given in compressed forms $c(T)$ and $c(P)$. They sought all appearances of P in T, without decompressing T. This achieves a search time that is sublinear in the size of the uncompressed text $|T|$. They showed that for the two-dimensional run-length compression there is a $O(|c(T)|\log|P|+|P|$, or almost optimal algorithm. The algorithm uses a novel multi-dimensional pattern matching technique - two-dimensional periodicity analysis.

A technique for Extending Rapid Exact-Match Spring Matching to Arrays of More than One Dimension by Theodre P Baker[20].

In this paper, a class of algorithms is presented for very rapid on-line detection of occurrences of a fixed set of pattern arrays as embedded subarrays in an

input array. By reducing the array problem to a string matching problem in a natural way, it is shown that efficient string matching algorithms may be applied to arrays. This is illustrated by use of the string-matching algorithm of Knuth, Morris and Pratt. Depending on the data structure used for the pre-processed pattern graph, this algorithm may be made to run "real-time" or merely in linear time. Extensions can be made to non-rectangular arrays, multiple arrays of dissimilar sizes, and arrays of more than two dimensions. Possible applications are foreseen to problems such as detection of edges in digital pictures and detection of local conditions in board games.

Efficient Pattern Matching with Scaling by Amihood Amir, Gad M Landau and Uzi Vishkin[21].

In this paper, the problem of pattern matching with scaling is defined. The input for two-dimensional version of the problem consists of an NxN "text" matrix and an MxM "pattern" matrix. Their goal was to find all occurrences of the pattern in the text, scaled to all natural multiples. That is, for every natural number $i, 1 \leq i \leq \left\lceil \frac{n}{m} \right\rceil$ they sought all occurrences of the pattern in the text, where each character of the pattern corresponds to an IxI square in the text. This problem is useful for some tasks in computer vision. Their main contribution was a linear time algorithm for the problem. Also considered situations where the text is provided in a less redundant form. For instance, suppose that a repeating character is compressed into one character, along with the number of repetitions. They showed how to enhance their algorithm so that its running time may become sublinear with respect to the original redundant input representation. Their algorithms where based on a new algorithmic approach to two-dimensional string-matching. Unlike existing

approaches, the new approach does not work by reducing a two-dimensional problem into a one-dimensional problem.

Two-Dimensional Periodicity and it's Applications by Amihood Amir and Gary Benson[22].

This paper presents a new algorithmic technique for two-dimensional matching, that of periodicity analysis. Periodicity in strings has been used to solve string-matching problems. The success of these algorithms suggests that periodicity could be as important as a tool in multi-dimensional matching.

This paper's main contribution was defining and analysing two-dimensional periodicity in rectangular arrays. In addition, they introduced a new pattern matching paradigm - *Compressed Matching*. A text array *T* and a pattern array *P* are given in compressed form *c(T)* and *c(P)*. They sought all appearances of *P* in *T*, without decompressing *T*. By using periodicity analysis, they showed that for two-dimensional run-length compression there is a $O(|c(T)|\log|P|+|P|$, or almost optimal algorithm that can achieve a search time that is sublinear in the size of the text $|T|$.

A novel Moment-Based Shape Description and Recognition Technique by H K Sardana, M F Daemi, A Sanders and M K Ibrahim[23].

In this paper, the invariance to rotation and translation is achieved with the use of global techniques such as moments, Fourier descriptors and the cyclic chain codings of polygonal approximation of the objects. The scale invariance is

also reported to be achieved with some additional computational cost. Such final description is termed as an n-dimensional feature vector represented as a point in n-dimensional space. Minimum-distance object classification can efficiently be used with such a description.

2.5. Fractal Signals and Images Paper

In this section, we represents samples of the most important of the outcome from the previous research work papers related to Fractal Image Data Compression, which some of these papers have been implemented in building up the DIP library package.

A Guided Tour of the Fractal Image Compression Literature by Diertmar Saupe and Raouf Hamzaoui[24].

Since the conception of fractal image compression by M.F. Barnsley a round 1987 the research literature on this topic has experienced a rapid growth, which is hard to keep track of. This paper contains a brief description of the major advances in the field. The corresponding relevant papers are presented in the form of original (slightly edited) abstracts. Also included is a comprehensive bibliography, the largest published on this topic to this date.

On the Synthesis and Processing of Fractal Signals and Images by Jonathan M. Blackledge[25].

This paper discussed some of the techniques available for synthesising and processing random fractal signals and images. The methods presented are derived from a Fourier-based description of a random scaling fractal and are

therefore able to utilise a Fast Fourier Transform. This provides the potential for constructing a real time facility by implementing available DSP hardware, the principal criterion for developing the techniques presented in this paper.

Method and Apparatus for Processing Digital Data by Barnsley et al[26].

Digital image data is automatically processed by dividing stored image data into domain blocks and range blocks. The range blocks are subjected to processes such as a shrinking process to obtain a mapped range blocks. Then, for each domain block, the mapped range block which is most similar to the domain block is determined and the address of that range block and the processes the block was subjected to are combined as an identifier which is appended to a list of identifiers for other domain blocks. The list of identifiers for all domain blocks is called a fractal transform and constitutes a compressed representation of the input image. To decompress the fractal transform and recover the input image, an arbitrary input image is formed into range blocks and the range blocks processed in a manner specified by the identifiers to form a representation of the original input image. This particular paper was the basic idea for coding the Fractal Transform Image Compression. This compression algorithm was the key solution to solve the rotation variant problem in pattern recognition.

Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations by Arnaud E. Jacquin[27].

In this paper, they proposed an independent and novel approach to image coding, based on a fractal theory of iterated transformations. The main characteristics of this approach are that

i) It relies on the assumption that image redundancy can be efficiently exploited through *self-transformability* on a blockwise basis, and

ii) It approximates an original image by a *fractal image*.

They therefore referred to their approach as *fractal block coding*. The coding-decoding system is based on the construction, for an original image to encode of a specific image transformation a fractal code which, when iterated on any initial image, produces a sequence of images which converges to a fractal approximation of the original.

A Review of Iterated Function System Theory for Image Compression by J. Waite[28].

In this paper they presented a tutorial introduction to the theory of iterated function systems (IFSs) as applied to digital images and digital image compression. There have been a number of excellent mathematical accounts of IFS theory applied to images but their aim in this paper was to show, using only elementary methods, why the IFS approach to digital image compression works. The paper described how an arbitrary image can be encoded using an IFS but they will not concentrated on the implementation of particular algorithms, descriptions of which can be found elsewhere.

Image Data Compression Using Fractal Techniques by J.M. Beaumont[29].

This paper introduced fractals and explained how affine transforms can be used to generate fractal pictures. It goes on to discuss the properties of affine transforms in more detail and how they represent a compressed version of

fractal pictures. A method of decomposing a general signal into affine transforms was outlined, and how this method has been extended and refined to compress visual image data.

Algorithm for Fast Image Compression by John Kominek[30].

This paper introduces the Fast Fractal Image Compression algorithm, a new approach to breaking the "speed problem" that has plagued previous efforts. For still images, experiments show that at comparable quality levels the FFIC algorithm is 5 to 50 times faster than the current state of the art. Such an improvement brings real-time video applications within the reach of fractal mathematics.

Three-Dimensional Fractal Video Coding by Kai Uwe Barthel and Thomas Voye[31].

In this paper, they proposed an extension to three-dimensional fractal video coding with spatial and temporal contraction in the 3D-frequency domain. High coding efficiency is achieved with an entropy constrained selection of the fractal transform parameters. As the new coding scheme uses no motion compensation the computational complexity can be kept low. Compared to two-dimensional coders an improved approximation quality is obtained.

Adaptive Fractal Image Coding in the Frequency Domain by Kai Uwe Barthel and Thomas Voye[32].

A block-based fractal image coder is able to exploit the redundancy of grey level images by describing image blocks through constructively transformed

blocks of the same image. Previous fractal coders used affine linear transformation in combination with 1st order luminance transformations that change the brightness and scale the luminance values of image blocks. They proposed an extension to high order luminance transformations that operate in the frequency domain. With this transformation and an adaptive coding scheme a better approximation of image blocks can be achieved. Bitrate reductions are higher than those achieved by "spatial-domain" fractal coding schemes. An additional effect of this new transformation is a better convergence at the decoder.

Hierarchical Interpretation of Fractal Image Coding and Its Application to Fast Decoding by Z.Baharav, D. Malah and E. Karnin[33].

This paper described the basics of a block oriented fractal image coder. The output of the coder is an IFS code, which describes the image as a fixed-point of a constrictive transformation. A new hierarchical interpretation of the IFS code, which relates different scales of the fixed-point to the code, is presented and proved. The proof is based on finding a function of a continuous variable, from which different scales of the signal can be derived. Its application to a fast decoding algorithm is then described, leading typically to an order of magnitude reduction of computation time.

A new Image Coding Technique Unifying Fractal and Transform Coding by Kai Uwe Barthel, S. Jorg and Thomas Voye[34].

They presented a new image coding scheme based on an unification of fractal and transform coding. They introduced a generalisation of the luminance transformation used by fractal coding schemes. By extending the luminance

transformation to the frequency domain, fractal and transform coding becomes subsets of the proposed transformation. Their new coding scheme FTC combines the advantages of both techniques. The Compared to JPEG a coding gain of 1.5 - 2.5 dB[PSNR] is obtained. The encoding time is reduced compared to conventional fractal coding schemes and a better convergence at the decoder is attained. At equal error rates the subjective quality of images coded with the new scheme is superior compared to transform coded images.

Analysis, Generation and Compression of Pavement Distress Images Using Fractals by Maaruf Ali, Michael A. Gennert and Trevor G. Clarkson[35].

This paper discusses the use of fractal to analyse, compress and generate pavement distress features (cracks). A method for calculating the fractal dimension is presented and values for pavement (cracks) reported. Several methods for fractal image compression are explained, especially the midpoint displacement algorithm to generate pavement distress images and IFS codes. The use of fractal techniques to generate standard images for testing an automated surface distress evaluation system is proposed.

Survey of Block Based Fractal Image Compression and Its Applications by Maarfuf Ali and Trevevor G. Clarkson[36].

A survey of block based fractal image compression and decompression algorithms are presented and its applications. Comparisons are made with various discrete cosine transform(DCT) algorithms.

The Use of Fractal Theory in a Video Compression System by Maarfuf Ali and Trevevor G. Clarkson[37].

The paper describes how Fractal Coding Theory may be applied to compress video images using an image resembling sequencer(IRS) in a video compression system on a modular image processing system.

2.6. Discussion

All the samples of the previous research work papers presented in the previous sections give an idea to the work that has been done in various areas. The research work in this thesis is reflected by the order of the sections starting with different techniques for image processing, such as filtering, rotating, segmenting, noise-reduction, etc. all of which have been designed from scratch and added to the DIP software library.

There are many techniques (Adrian 1991, Michael 1988, William 1978 and others) for image processing and it is important to know that almost all image processing techniques are based on clear and unambiguous paradigms .

The second part of this chapter is concerned with Artificial Neural Networks (ANN). This is explained in greater detail in Chapter 4. The papers in this chapter (section 2.2) address the problems of object pattern recognition and classification (Roberts 1992, Phillip 1993 and Luis 1989). They are used to train multi-layer ANNS using either Hopfield or Back-Propagation Algorithms (Suganthan 1995, Feug, Houkesand and Spreeuwers 1992) using pre-processed image (segmented data) or using advance methods to extract image feathers.

Pattern recognition is one of the oldest problems in image processing and indicated the work of psychologists. The theories of psychologists led the other of ideas that give birth to early neural network research. A very important point to make about ANN is that it takes a long computing time and a large memory space to solve a problem. The third part of this chapter (section 2.3) is dealing with image data compression methods. The key papers have been given, this section addresses a common problem in modern technology, that of reducing the image data file which is a central topic of this thesis. There are a number of different image compression techniques used to reduce the image data file (Edward uses a two-level one-bit non-parametric quantifier, Burt and Adelson uses the mean formula). The main problem with data image compression is how to reconstruct the compressed image without losing any of its features.

The fourth section of this chapter is about pattern matching and pattern recognition without using ANN. Matching or recognition using primitive axiomatic ideas (Baker uses string matching, Amir, Vishkin and Landan use matching with different scales) and (Amir and Benson use matching by pattern representation over 360 degrees in increments by 90 degrees). Rotation in increments of 90 degrees eliminates the approximations due to rotation, but also limits the problem of space.

The final part of this chapter (section 2.5) is about Fractals. This section gives in brief the Fractal theory, generations and its applications, such that fractal compression methods, computing the fractal dimension and advanced fractal algorithms. The main advantages of using the fractal compression technique is that it gives the highest compression ratio of all the existing compressors, and gives an identical decompressed image, which make these type of compressors

reliable. The IFS approach is the heart of the fractal compressors and it's work for the most advantage new fractal compression algorithms (first introduced by M. Barnsley in 1988). The first generation of fractal compression was slow. There are many research projects to improve the speed of these algorithms (John Kominer introduced one), his research is focusing in the search part of the fractal algorithm. The Fractal Image Transform Compression Technique was the key research points in this thesis and has been used as a method of solving the invariant rotation and scaling pattern recognition problems.

References

References

[1] **Thomas S. Hung, George J Yang and Gregory Y. Tang,** *A Fast Two-Dimensional Median Filtering Algorithm*, IEEE Digital Image Processing, Volume 1, May 1987, pp 66-79.

[2] **Anil K. Jain**, *Advances in Mathematical Models for Image Processing*, IEE Digital Image Processing, Volume 1, May 1987, pp 114-129.

[3] **Jong-Sen Lee**, *Digital Image Enhancement and Noise Filtering by Use of Local Statistics*, IEEE Digital Image Processing, Volume 1, May 1987, pp212-225.

[4] **A.Roberts, M.Yearworth**. *Comparison of pre-processing Transforms for Neural Network Classification of Character Images*, IEEE. The Fourth International Conference on Image Processing and its Applications, Vol. 7-9, April 1992, pp 189-200.

[5] **Philip R. Nachtsheim,** *A Stable Second Order Method for Training Back Propagation Networks*; AIAA, Information Sciences Division (NASA ARC). Moffett Field, CA 94035, 1993.

[6] **Luis B. Almeida, R. Alves Redol,** *Back Propagation in Perceptrons with Feedback*; Lisboa, pp 2-9.

[7] **P N Suganthan, E K Teoh and D P Mital**, *Pattern Recognition by Homomorphic Graph Matching using Hopfield Neural Networks*, Image and Vision Computing; Volume 13, Number 1, Feb. 1995, pp 45-59.

[8] **A K Muhamed and F Deravi,** *Neural Networks for Texture Classification*, IEEE 4th Conference, Image Processing and its Applications, April 1992, pp 201-204.

[9] **E C Mertzanis, D A Dukic, I D Benest and J Austin,** *A Neural Network Based Position Invariant Pattern Recognition System with a Constrained Hypermedia Style User-Interface*, IEEE, 4th Conference, Image Processing and its Applications, April 1992, pp 217-220.

[10] **Tian-Jin Feng, Z Houkes, M J Korstan and L J Spreeuwers,** *Internal Measuring Models in Trained Neural Networks for Parameter Estimation from Images*, IEEE 4th Conference, Image Processing and its Applications, April 1992, pp 230-233.

[11] **Anil K Jain**, *A Fast Karhunen-Loeve Transform for a Class of Random Processes*, IEEE Transactions on Communications; September 1976.

[12] **Harry C Andrews, Claude L Patterson**, *Singular Value Decomposition (SVD) Image Coding*, IEEE Transactions on Communication, September 1976.

[13] **Edward J Delp, Rangasami L Kashyap and O Robert Mitchell**, *Image Data Compression Using Autoregressive Time Series Models*; IEEE Pattern Recognition; Vol. 11, pp 305-315.

[14] **James W Modestino, V Bhaskaran**, *Robust Two-Dimensional Tree Encoding of Image*, IEEE Transactions on Communication, December 1981.

[15] **Edward J Delp, O Robert Mitchell,** *Image Compression Using Block Trunction Coding*, IEEE Transactions on Communication, September 1979, pp 329-336.

[16] **Peter J Burt, Edward H Adelson,** *The Laplacian Pyramid as Compact Image Code*, IEEE Transactions on Communication, April 1983.

[17] **Donald J Healy, O Robert Mitchell,** *Digital Video Bandwidth Compression Using Block Trunction Coding*, IEEE Transactions on Communication; December 1981.

[18] **C A Papadopoulos and T G Clarkson,** *Data Compression of Moving Images Using Geometric Transformations*, IEEE The 4th Conference Image Processing and its Applications; April 1992, .

[19] **Amihood Amir and Gary Benson**, *Efficient Two-Dimensional Compressed Matching*; IEEE Data Compression Conference; March 1992.

[20] **Theodre P Baker**, *A technique for Extending Rapid Exact-Match Spring Matching to Arrays of More than One Dimension*, SIAM J Compute; Vol. 7, No 4; November 1978.

[21] **Amihood Amir, Gad M Landau and Uzi Vishkin**, *Efficient Pattern Matching with Scaling*, ACM-SIAM SYMP on Discrete Algorithms; part 1, January 1990, pp 344-357.

[22] **Amihood Amir, Gary Benson**, *Two-Dimensional Periodicity and it's Applications*, ACM-SIAM Symp on Discrete Algorithms; part 1; January 1990, pp 440-453.

[23] **H K Sardana, M F Daemi, A Sanders and M K Ibrahim,** *Anoval Moment-Based Shape Description and Recognition Technique*, IEEE the 4th Conference Image Processing and it's Applications; April 1992.

[24] **Diertmar Saupe and Raouf Hamzaoui,** *A Guided Tour of the Fractal Image Compression Literature*, University of Freiburg, Germany.

[25] **Jonathan M. Blackledge**, *On the Synthesis and Processing of Fractal Signals and Images,* 1992, pp 445-450.

[26] **Barnsley et al,** *Method and Apparatus for Processing Digital Data*, United States Patent No 5065447.

[27] **Arnaud E. Jacquin,** *Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations*, IEEE Transactions on Image Processing, Vol. 1. No 1, January 1992, pp 18-29.

[28] **J. Waite,** *A Review of Iterated Function System Theory for Image Compression, pre-print.* British Telecom Research Labourites, Martlesham Heath, UK (1992).

[29] **J.M. Beaumont,** *Image Data Compression Using Fractal Techniques*, BT Technol, Vol. 9, No 4, October 1991, pp 93-109.

[30] **John Kominek,** *Algorithm for Fast Image Compression.* Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

[31] **Kai Uwe Barthel and Thomas Voye,** *Three-Dimensional Fractal Video Coding* , University of Berlin, D-10587 Berlin, Germany.

[32] **Kai Uwe Barthel and Thomas Voye,** *Adaptive Fractal Image Coding in the Frequency Domain*, University of Berlin, D-10587 Berlin, Germany, pp 1-10.

[33] **Z.Baharav, D. Malah and E. Karnin**, *Hierarchical Interpretation of Fractal Image Coding and Its Application to Fast Decoding*, Israel Institute of Technology, Department of Electrical Engineering, Haifa 32000, Israel.

[34] **Kai Uwe Barthel, S. Jorg and Thomas Voye,** *A new Image Coding Technique Unifying Fractal and Transform Coding*, University of Berlin, D-10587 Berlin, Germany, pp112-116.

[35] **Maaruf Ali, Michael A. Gennert and Trevor G. Clarkson**, *Analysis, Generation and Compression of Pavement Distress Images Using Fractals*, King's College London, Department of Electronic and Electrical Engineering, London, England, pp 228-259.

[36] **Maaruf Ali, Michael A. Gennert and Trevor G. Clarkson,** *Survey of Block Based Fractal Image Compression and Its Applications*, King's College London, Department of Electronic and Electrical Engineering, London, England, pp 1-13.

Chapter 3

Digital Image Processing

This chapter explains what DIP is, what it is concerned with and gives the basic theory of the DIP algorithms used in this thesis.

In this chapter **Page No**

3. Digital Image Processing

3.1. Introduction to Image Processing

Image processing is a science that deals with images data. It covers a broad spectrum of techniques that are applicable to a wide range of applications. Image processing can be thought of as a special form of two-dimensional processing used to uncover information about images. In general, image processing techniques are applied to images or image data where:

a. Enhancement or modification of an image is necessary to improve appearance or to highlight some aspect of the information contained in the image.

b. Elements within an image need to be categorised, classified, matched, or measured.

c. Portions of images need to be combined or image elements need to be re-organised.

The techniques of image processing can be applied to data even if the data is not in a visible form. The manipulation of visible image data is just one of the many uses of image processing - probably the predominant one. Image processing can be used to produce a visible image of purely numeric data enhanced in some manner to highlight some aspect of the data. Examples of this kind of image processing can be found in magnetic resonance medical imaging equipment, sonar, radar, ultrasound equipment, heat sensing equipment, fractals and so on. Applying an image processing algorithm to an image is not always done with the appearance of the image in mind. In fact, the result might not be pleasing to look at. Aesthetics are not the only criterion by which to judge the effectiveness of the applied transformation. If the transformation is designed to bring out additional information and/or

details not visible in the original image, the result can be considered successful even if it is not pleasing to look at.

3.2. What a digital image is

A digital image can be considered to be a visual display of a matrix of (integer) numbers whose values determine particular shades of grey (for grey level images), or specific colours (colour images). The matrix or array represents picture (image) elements or pixel. The pixel may be black and white or various shades of grey. If there are two hundred and fifty six possible shades of grey, then the image can be completely described in digital terms by a binary string of 128x128 binary digits (8 bits) each having a value of one or zero. Such a representation of an original scene is known as "digitized" image and is suitable for processing by a digital computer. ***Mathematically***, a grey level image can be represented by a function of two variables f(x, y) which gives a number z = f(x, y) corresponding to a grey level at the point (x, y). A grey level digital image can therefore be considered to be a discrete function.

$$z_k = f_{ij} \text{ where } f_{ij} \equiv f(x_i, y_j)$$

Here, f_{ij} is the value of the function at $x = x_i$ and $y = y_j$

3.3. Digitisation

Digitisation is the process of converting an image into an 2D array or 'matrix' of numbers.

There are two components to digitisation:

1. Sampling or Spatial Quantization
2. Grey level or Luminance Quantization

3.3.1. Spatial Quantization

Spatial quantization corresponds to sampling the brightness of the image at a number of points - usually a rectangular grid. Spatial quantization gives rise to an array of numbers which can be taken to be an approximation to the original image.

f_{ij} approximates f(i,j)

Each element of the array f_{ij} is referred to as a pixel - a picture element.

3.4. The Sampling Theorem

An analogue image can be reconstructed exactly from its digital form as long as the sampling frequency (i.e. samples per linear measure) is at least twice the highest frequency present in the image.

$$\text{Sampling interval} \leq \frac{1}{\text{Nyquist frequency}}$$

where Nyquist frequency = 2 x (Maximum frequency of image).

3.5. Mathematical model for a digital image

Data = (Point Spread Function) Convolve (Object Function) + Noise

- **Point Spread Function:** Describes the way information on the object function is spread as a result of recording the data. It is a characteristic of the imaging instrument and is a deterministic function.

- **Noise:** A non-deterministic function which can at best only be described in terms of some statistic. Noise is a stochastic function which is a consequence of all the unwanted external disturbances that occur during the recording of the data.

- **Object function:** Describes the object that is being imaged - its surface or internal structure for example.

- **Convolution:** A mathematical operation which 'smears' one function with another.

3.6. Principal elements in Digital Image Processing

- **Image restoration and reconstruction:** Concerned with operations which aim to recover information about an object from data which is blurred and corrupted by noise.

- **Image enhancement:** Concerned with processes whose goal is to improve the 'quality' or 'readability' of an image in some ways to emphasise features of particular importance or relevance.

- **Pattern recognition:** Concerned with the identification or interpretation of an image. It aims to extract the high level information which the image is intended to convey.

- **Image understanding:** Attempts to interpret an image in terms of the 'physics' of the imaging system.

3.7. Computational background

The computational background to DIP involves a number of techniques of numerical analysis. All the techniques are using the standard moving window in which the window is moved one element at a time over an image, a set of values are obtained as a function of the window position. The standard moving window technique can be used by two methods:

(i) Without border which is move the window inside the image frame.

(ii) With border which is add another frame around the image frame so that the additional frame will be used as a new border for the image.

Those techniques which are of particular value are:

- **Solutions to linear systems of equations**
- **Finite difference analysis**

Many DIP algorithms can be classified in terms of a digital filter. There are two important classes of digital filter used in DIP.

- **Convolution filters**

Convolution filters are non-recursive filters. They are linear processes which operate on the data directly.

- **Fourier filters**

Fourier filters operate on data obtained by computing the Discrete Fourier Transform of an image. This is accomplished using the Fast Fourier Transform algorithm.

3.7.1. Digital Filters

Digital filters fall into two main categories:

1. Real space filters
2. Fourier space filters

3.7.2. Real space filters

Real space filters are based on some form of 'moving window' principle. A sample of data about a given element of the image is processed giving (typically) one output value. The window is then moved on to the next element of the image and the process is repeated. A common real space filter is the **Finite Impulse Response** or **FIR** filter. This is a non-recursive filter.

3.7.2.1. Discrete Convolution in 2D

$$S_{ij} = P_{ij} \otimes\otimes f_{ij} = \sum_{n} \sum_{m} P_{i-n,j-m} f_{nm}$$

3.7.2.2. Discrete Correlation in 2D

$$S_{ij} = P_{ij} \circ\circ f_{ij} = \sum_{n} \sum_{m} P_{n-i,m-j} f_{nm}$$

3.7.3. Fourier space filters

Fourier space filters are usually multipliactive operations which operate on the Discrete Fourier Transform (DFT) of the image. If S_{ij} , P_{ij} and F_{ij} are taken to denote the DFT's of S_{ij}, P_{ij} and F_{ij} respectively, then in Fourier space,

$$S_{ij} = \sum_{n} \sum_{m} P_{i-n,j-m} f_{nm}$$

transforms to

$$S_{ij} = P_{ij} F_{ij}$$

This is known as the (discrete) convolution theorem. If P_{ij} is composed of just a few elements, then the discrete convolution can be computed directly. If P_{ij} is composed of many elements, then it is numerically more efficient to use a Fast Fourier Transform (FFT) and perform the filtering operation in Fourier Space.

3.7.3.1. The Fourier Transform

$$f(x,y) = \frac{1}{(2\pi)^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(K_x, K_y) \exp(iK_x x) \exp(iK_y y) dK_x dK_y$$

$$F(K_x, K_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) \exp(-iK_x x) \exp(-iK_y y) dx dy$$

$F(K_X, K_Y)$ is the Fourier transform of $f(x,y)$ - a non-periodic function.

3.7.3.2. The Discrete Fourier Transform (DFT)

$$f_{pq} = \frac{1}{N^2} \sum_n \sum_m F_{nm} \exp(2\pi inp/N) \exp(2\pi imq/N)$$

$$F_{nm} = \sum_p \sum_q f_{pq} \exp(-2\pi inp/N) \exp(-2\pi imq/N)$$

3.7.3.3. The Fast Fourier Transform (FFT)

The FFT is an algorithm for computing the Discrete Fourier Transform with fewer additions and multiplications. The DFT (standard form) of a N-point vector is given by

$$F_m = \sum_n f_n \exp(-2\pi inm/N)$$

How much computation is involved in computing the DFT of N points?

Write $W_n = \exp(-2\pi i/N)$

Then $F_m = \sum_n W_n^{nm} f_n$

The above result is a matrix equation which can be written in the form

$$\begin{pmatrix} F_0 \\ \cdot \\ \cdot \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} W_N^{00} & \cdots & W_N^{0(N-1)} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ W_N^{(N-1)0} & \cdots & W_N^{(N-1)(N-1)} \end{pmatrix}$$

In this form, we see that the DFT is essentially computed by multiplying an N-point vector f_n by a matrix of coefficients given by a (complex) constant Wn to the power of nm. This requires N x N multiplications

Basic idea behind the FFT algorithms

By applying a simple but very elegant trick, a N-point DFT can be written in terms of two $\frac{N}{2}$ point DFTs. The FFT algorithm is based on repeating this trick again and again until a single point DFT is obtained.

Basic idea

$$\sum_{n=0}^{n-1} f_n \exp\left(-2\pi inm / N\right)$$

$$= \sum_{n=0}^{(N/2)-1} f_{2n} \exp\left(-2\pi i(2n)m / N\right) + \sum_{n=0}^{(N/2)-1} f_{2n+1} \exp\left(-2\pi i(2n+1)m / N\right)$$

$$= \sum_{n=0}^{(N/2)-1} f_{2n} \exp\left[-2\pi inm / (N/2)\right] + \exp\left(-2\pi im / N\right) \sum_{n=0}^{(N/2)-1} f_{2n+1} \exp\left[-2\pi inm / (N/2)\right]$$

$$= \sum_{n=0}^{(N/2)-1} f_{2n} W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} f_{2n+1} W_{N/2}^{nm}$$ Thus:

DFT of N-point array = DFT of even components + W_N^m x DFT of odd components.

Using the subscripts e and o to represent odd and even respectively, we can write this result in the form

$$F_m = F_m^e + W_N^m F_m^o$$

The important thing to note here, is that the evaluation of F_m^e and F_m^o is over N/2 points - the N/2 even components and the N/2 odd components of the original N-point array. To compute F_m^e and F_m^o we only need half the number of multiplications that are required to compute F_m. We can continue subdividing the data into odd and even components until we get down to the DFT of a single point. Because the data is subdivided into odd and even

components equal length we require an initial array of size $N=2^k$, K=1,2,3.... Computing the DFT in this way reduces the number of multiplications needed to $N \log_2 N$ which for even moderate values of N is considerably smaller than N^2.

3.8. The Spectrum of an Image

The DFT of an image is given by

$$F_{nm} = \sum_p \sum_q f_{pq} \exp\left(-2\pi inp / N\right) \exp\left(-2\pi mq / N\right)$$

Important results:

(i) Complex spectrum

$$F_{nm} = G_{nm} + iH_{nm}$$

$$\text{where } G_{nm} = R_e\left[F_{nm}\right] \text{and } H_{nm} = I_m\left[F_{nm}\right.$$

(ii) Amplitude spectrum

$$\left| F_{nm} \right| = \sqrt{G_{nm}^2 + iH_{nm}^2}$$

(iii) Phase spectrum

$$j_{nm} = \tan^{-1}\left(\frac{H_{nm}}{G_{nm}}\right)$$

(iv) Power spectrum

$$\left| F_{nm} \right|^2$$

Since an image is usually taken to be a positive function, its DC level is typically very much larger than the other components and swamps detail in the

rest of the spectrum. It is therefore difficult to observe and analyse a grey level or colour coded display of the spectrum. To overcome this problem, the DC component is often left out of the displays by setting it to zero after the DFT has been computed. Alternatively, a display can be represented based on an algorithmic scale, i.e. generating a display of

$$\ln\left(1+\left|F_{mn}\right|\right)$$

1 is added in case any zeros are present in.

3.9. Image Restoration and Reconstruction

All image information systems are inherently resolution limited. Images can become blurred due to a variety of physical effects such as motion in the object or image planes, the effects of turbulence and refraction and/or diffraction. One of the principal objectives of image restoration and reconstruction is to recover information about an object from data which is 'blurred' and corrupted by noise. Most of the techniques employed are either directly or indirectly based on a mathematical model for the blurred image which involves the convolution of two functions - the Point Spread Function and the Object Function. 'De-blurring' an image amounts to solving the inverse problem by this model which is known as 'deconvolution'. Image restoration attempts to provide a resolution compatible with the bandwidth of the imaging system (a resolution limited system). Image reconstruction attempts to provide a resolution which is greater than the resolution of the data provided (usually the resolution limit of the imaging system). This is known as super resolution.

3.9.1. General Statements

In general, there is no exact or unique solution to the image restoration/reconstruction problem - it is an ill-posed problem. We attempt to

find a 'best estimate' solution based on some physically viable condition. The imaging equation is given by

$$\mathbf{S} = \mathbf{p} \otimes\otimes \mathbf{f} + \mathbf{n}$$

or using the convolution theorem

$$\mathbf{S} = \mathbf{PF} + \mathbf{N}$$

Assuming that F is a broadband spectrum, there are two cases we must consider:

(i) $P(K_x, K_y) \rightarrow f$ as K_x and K_y approach ∞

3.9.1.1. Image restoration: Recover F given S.

(ii) $P(K_x, K_y)$ is bandlimited, i.e. $P(K_x, K_y) = 0$ for certain values of K_x and/or K_y.

3.9.1.2. Image reconstruction: Recover F given S - requires frequency components to be 'synthesised' beyond the bandwidth of the data. This is an extrapolation problem.

3.9.2. Restoration of Blurred Images using the Least Squares Principles

Statement of the problem: Given the imaging equation

$$\mathbf{S}_{ij} = \mathbf{P}_{ij} \otimes\otimes \mathbf{f}_{ij} + \mathbf{n}_{ij}$$

find an estimate for $\mathbf{f}_{ij}$, denoted by $\hat{\mathbf{f}}_{ij}$ such that

$$\mathbf{e} = \left\| \mathbf{f}_{ij} - \hat{\mathbf{f}}_{ij} \right\|^2$$

is a minimum.

We consider an estimate for fij of the form

$$\hat{\mathbf{f}}_{ij} = \mathbf{q}_{ij} \otimes\otimes \mathbf{s}_{ij}$$

The problem is then reduced to finding q_i or equivalently Q_{ij} such that e is a minimum, i.e.

$$\frac{\partial \mathbf{e}}{\partial \mathbf{q}_{ij}} = \mathbf{0}$$

Q_{ij} is called Wiener filter. It can be derived that the expression for the least squares or Wiener filter is:

$$\mathbf{Q}_{ij} = \frac{\mathbf{P}_{ij}}{\left|\mathbf{P}_{ij}\right|^2 + \left|\mathbf{N}_{ij}\right|^2 \left|\mathbf{F}_{ij}\right|^2}$$

From the algebraic form of the Wiener filter it is clear that this particular filter depends on:

(i) the functional form of the Point Spread Function $\left|P_{ij}\right|^2$ that is used;

(ii) the functional form of the Noise to Signal ratio $\left|N_{ij}\right|^2 / \left|F_{ij}\right|^2$.

In many practical cases when we do not have access to successive images and hence the functional form of the NSR we make an approximation and consider a Wiener filter of the form

$$\text{Wiener filter} = \frac{\mathbf{P}_{ij}}{\left|\mathbf{P}_{ij}\right|^2 + \textbf{constant}}$$

The constant ideally reflects any available information on the average Signal to Noise ratio of the image, i.e.,

$$\text{Constant} = \frac{1}{(\text{SNR})^2}$$

where SNR stands for signal to noise ratio. In practice, the exact value of this constant must be chosen by the user. Before attempting to deconvolve an image the user must at least have some *a priori* knowledge on the functional form of the Point Spread Function. Absence of this information leads to a method of approach known as **blind deconvolution**. A common technique is to assume that the Point Spread Function is Gaussian, i.e.,

$$P_{ij} = \exp\left[-\left(i^2 + j^2\right)/2\sigma^2\right]$$

where σ is the standard deviation which must be defined by the user.

In this case, the user has control of two parameters:

(i) The standard deviation of the Gaussian Point Spread Function;

(ii) The Signal to Noise Ratio.

In practice, the user must adjust the parameters until a suitable 'user optimised' reconstruction is obtained. In other words, the Wiener filter must be 'tuned' to give a result which is acceptable based on the judgement and intuition of the user.

3.9.3. Restoration of Images using the Maximum Entropy Principle

Basic idea: Given the imaging equation

$$S_{ij} = P_{ij} \otimes\otimes f_{ij} + n_{ij}$$

find f_{ij} such that the general definition of the entropy E defined by

$$E = \sum_i \sum_j p_{ij} \ln p_{ij}$$

is a maximum. It can be proved that the linearized maximum entropy filter has the form

$$\frac{P_{ij}}{|P_{ij}|^2 + 1/2\lambda}$$

where the constant λ is called the **Lagrange multiplier**. This filter is very similar to the Wiener filter. The only difference is that the Wiener filter is regularised by a constant determined by the SNR of the data whereas this filter is regularised by a constant determined by the Lagrange multiplier.

3.9.4. Restoration of Images using the Power Spectrum Equation Principle

Given the imaging equation

$$S_{ij} = P_{ij} \otimes\otimes f_{ij} + n_{ij}$$

find an estimation $\hat{f}_{ij}$ whose power spectrum is equal to the power spectrum of the desired function f_{ij}. $\hat{f}_{ij}$ is obtained by employing the criterion

$$|F_{ij}|^2 = |\hat{F}_{ij}|^2$$

together with the linear convolution model

$$\hat{f}_{ij} = q_{ij} \otimes\otimes S_{ij}$$

The PSE filter has the form

$$\text{PSE filter} = \left(\frac{1}{|P_{ij}|^2 + \frac{|N_{ij}|^2}{|F_{ij}|^2}} \right)^{1/2}$$

Like the Wiener filter in the absence of accurate estimates for $|N_{ij}|^2$ and $|F_{ij}|^2$ we approximate the PSE by

$$\text{PSE filter} = \left(\frac{1}{|P_{ij}|^2 + \text{constant}} \right)^{1/2}$$

where

$$\text{constant} = \frac{1}{(\text{SNR})^2}$$

3.9.5. Reconstruction of Bandlimited Images using the Least Squares Principle

A bandlimited function is a function whose spectral bandwidth is finite. Most real signals and functions are bandlimited functions. This leads one to consider the problem of how the bandwidth, and hence for resolution of a bandlimited image, can be increased synthetically using digital processing techniques. In other words, how can we extrapolate the spectrum of a bandlimited function from an incomplete sample? The type of resolution that is obtained by spectral extrapolation is referred to as **super resolution**. An important aspect of practical solution to the spectral extrapolation problem is the incorporation of *a priori* information on the structure of an object.

3.9.6. The Gerchberg-Papoulis Method

Consider the case where we have an object f(x,y) characterised by a discrete spectrum F_{nm} which is composed of a finite number of samples:

$$-\frac{N}{2} \leq n \leq \frac{N}{2}$$

$$-\frac{M}{2} \leq m \leq \frac{M}{2}$$

These data are related to the signal by the equation

$$F_{nm} = \int_{-x}^{x}\int_{-y}^{y} f(x,y)\, e^{-i(K_n x + K_n y)}\, dxdy$$

where f is assumed to be finite support X and Y i.e.,

$$|x| \leq X \text{ and } |y| \leq Y$$

and K_n, K_m are discrete **spatial frequencies** (Nyquist samples of the analogue spectrum F).

Using this data, we can define the **B**and **L**imited function

$$F_{BL}(x,y) = \sum_{n}\sum_{m} F_{nm}\, e^{i(K_n x + K_m y)}$$

which is related to F_{nm} by a two dimensional Fourier Series. **The Problem:**

To reconstruct f given F_{nm} or equivalently, F_{BL}.

3.9.7. Least Square Analysis

Consider a model for an estimate $\hat{f}$ of f given by

$$\hat{f}(x,y) = \sum_{n} \sum_{m} A_{nm} e^{i(K_n x + K_m y)}$$

This model is just a two-dimensional Fourier Series representation of the object.

Given the model for $\hat{f}$ above, our problem is reduced to that of finding the coefficients A_{nm}. Using the least squares method, we compute A_{nm} by minimising the mean square error.

$$e = \int_{-x}^{x} \int_{-y}^{y} \left| f(x,y) - \hat{f}(x,y) \right|^2 dxdy$$

In this way what we get is:

$$F_{pq} = 4XY \sum_{n} \sum_{m} A_{nm} \, \mathrm{Sinc}\left[(K_p - K_n)X\right] \mathrm{Sinc}\left[(K_q - K_m)Y\right.$$

where

$$\mathrm{Sinc}(x) = \frac{\mathrm{Sin}(x)}{x}$$

The estimate $\hat{f}(x,y)$ can be computed by solving the equation above for the coefficients A_{nm}.

Note, the Sinc functions (in particular the zero locations) are sensitive to the precise values of X and Y and hence small errors in X and Y can dramatically effect the computation of A_{nm}.

In other words this equation is ill-conditioned. This (least squares) approach is known as the Gerchberg-Papoulis method.

3.9.8. Incorporation of a *priori* Information

Since we have considered an image f of finite support, we can consider the model for the estimation $\hat{f}$ of f to be given by:

$$\hat{f}(x,y) = W(x,y)\sum_{n}\sum_{m} A_{nm} e^{i(K_n x + K_m y)}$$

where

$$W(x,y) = \begin{cases} 1, & |x| \le X, \ |y| \le Y \\ 0, & |x| > X, \ |y| > Y \end{cases}$$

Writing the estimation $\hat{f}$ in this way, we observe that W (i.e. essentially the values of X and Y) represents a simple but crucial form of a *priori* information. Crucial, because this information is require to compute the Sinc functions and hence the coefficients A_{nm}.

The algebraic form of the above equation suggests incorporating further a *priori* information into the 'weighting function' W in addition to the support of the object f. We consider a model for $\hat{f}$ of the form

$$\hat{f}(x,y) = W(x,y)\sum_{n}\sum_{m} A_{nm} e^{i(K_n x + K_m y)}$$

where W is now a generalised weighting function composed of limited a *priori* information on the structure of f. We introduce a modified version of the least square model which involves minimising the error.

$$e = \int_{-x}^{x}\int_{-y}^{y} \left| f(x,y) - \hat{f}(x,y) \right|^2 \frac{1}{w(x,y)} \, dxdy$$

In this case we obtain the simple algebraic result

$$\hat{f}(x,y) = \frac{W(x,y)}{W_{BL}(x,y)} f_{BL}(x,y)$$

Here W_{BL} is a bandlimited function, bandlimited by the same extent as f_{BL}. The method presented above is a variation of the Gerchberg-Papoulis method.

The weighting function w(x,y) can be used to encode as much information as is available on the spatial characteristics of f(x,y). We can summarise this algorithm in the form

$$\text{reconstruction} = \frac{\text{bandlimited image x a } \textit{priori} \text{ information}}{\text{bandlimited a } \textit{priori} \text{ information}}$$

The success of this algorithm depends on the quality of the a *priori* information that is available, just as the performance of the Wiener filter or MEM depends upon a *priori* information on the functional form of the Point Spread Function.

3.10. Sinc Interpolation

Suppose we sample a function at regular intervals of S_x and S_y. The sampled function g is given by

$$g(x,y) = f(x,y)\sum_{n=-\infty}^{\infty}\sum_{m=-\infty}^{\infty} S(x - nS_x)S(y - mS_y)$$

Can be proved that, in Fourier space, this equation becomes

$$G(K_x,K_y) = \frac{2\pi}{S_x}\frac{2\pi}{S_y}\sum_n\sum_m F\left(K_x - \frac{2\pi n}{S_x}, K_y - \frac{2\pi m}{S_y}\right)$$

This result demonstrates that sampling the image f, creates a new spectrum G which is a periodic of the spectrum F spaced at regular intervals. Assuming that f has been sampled at the Nyquist frequency, the only difference between g and f is that the spectrum of g consists of F repeated at regular intervals.

$$\pm\sqrt{\left(\frac{2\pi n}{Sx}\right)^2\left(\frac{2\pi m}{Sy}\right)^2}\quad ;\ n = \pm1,\pm2,\pm3,\ldots,\pm\infty, m = \pm1,\pm2,\pm3,\ldots,\pm\infty$$

f can be obtained from g by retaining just the part of G for values of $\left|\vec{K}\right|$ less than or equal to K and setting all other values in the spectrum to zero i.e.

$$F\left(\vec{K}\right) = G\left(\vec{K}\right)$$

provided we set

$$G\left(\vec{K}\right) = 0\ \forall\left|\vec{K}\right| > K$$

We can describe this mathematically by multiplying G with the function

$$H\left(\vec{K}\right) = \begin{cases} 1, & \vec{K} \le K \\ 0, & \vec{K} > 0 \end{cases}$$

Thus, F is related to G by the equation

$$F\left(\vec{K}\right) = H\left(\vec{K}\right)\ G\left(\vec{K}\right)$$

Can be proved that

$$f(x,y) = \mathrm{Sinc}\left(k\sqrt{x^2+y^2}\right) \otimes\otimes\ g(x,y)$$

The restoration of a continuous function from a sampled one is known generally as interpolation. The above result shows that a function can be interpolated by convolving it with the appropriate sinc function. This is known as sinc interpolation.

3.11. Image Enhancement

The goal of image enhancement is literally to make an image look better. Image enhancement is a rather subjective matter because what 'looks better' depends on the type of details and contrasts in the image the user is hoping to acquire.

3.11.1. Simple Transforms

Simple transformations can be used to enhance details in an image which occur in dark or light regions of the image.

3.11.1.1. Logarithmic Transform

The logarithmic transform is of the general form

$$u_{out}(i,j) = \ln\left[1 + u_{in}(i,j)\right]$$

where u is the value of a pixel at a location (i,j). The portion of the logarithmic curve over which this transform is required can be adjusted by introducing a scaling parameter and employing the transform.

$$u_{out}(i,j) = \frac{1}{a}\ln\left[1 + \left(e^{a} - 1\right)u_{in}(i,j)\right]$$

Note: the addition of 1 is included to prevent problems occurring if $u_{in} = 0$.

The effect of this transform is to increase the dynamic range of dark regions in an image and decrease the dynamic range of light regions. The logarithmic transform is a relatively simple method of adjusting the visual quality of an image in favour of those factors which have low grey level values.

3.11.1.2. Exponential Transform

The exponential transform is the inverse of the logarithmic transform and has an inverse effect on the image, i.e. it enhances detail in the light regions of the

image while decreasing the dynamic range of grey levels in dark regions of the image. The basic transform is

$$u_{out}(i,j) = \exp\left[u_{in}(i,j)\right]$$

Bases other than the exponential can also be used for this purpose by employing the transform

$$u_{out}(i,j) = \frac{1}{a}\left[(1+a)^{u_{in}(i,j)} - 1\right]$$

where $\propto$ is a variable scaling parameter defined by the user.

3.11.2. Histogram Equalisation

Another way of enhancing an image is to apply a transform which modifies the histogram of the image in a pre-determined and desired fashion. A histogram is just a plot of the number of times a particular grey level occurs against the grey level. This provides a global description of the appearance of the image in terms of the characteristic probability distribution function of grey level. The profile or 'shape' of the histogram describes the density of grey levels in the image. If the histogram peaks at a low grey level, then the image is relatively dark, and if the histogram has a concentration of pixels with a high grey level, then the image is relatively light. Histogram equalisation attempts to generate an image whose histogram is uniform (i.e. an image where each grey level occurs an equal number of times: hence the term 'equalisation'). The effect is to increase the dynamic range of grey levels. This has a considerable influence on the visual appearance of the image, enhancing the details of many features in both the dark and light regions of the image.

3.11.3. Highpass Filtering

Highpass filters can be used to enhance the characteristics of an image governed primarily by the high frequency content of an image.

3.11.3.1. Ideal Highpass Filter

The ideal highpass filter is given by

$$H(K_x, K_y) = \begin{cases} 0, & \sqrt{K_x^2 + K_y^2} \le K \\ 1, & \sqrt{K_x^2 + K_y^2} > K \end{cases}$$

where K is the cut-off distance measured from the origin (the point in Fourier space where Kx = Ky = 0). This filter attenuates completely (i.e. sets to zero) all those spatial frequencies that are less than or equal to the cut-off frequency and passes without modification all those spatial frequencies that are greater than the cut-off frequency.

3.11.3.2. The Butterworth Highpass Filter

The Butterworth highpass filter (BHPF) is an approximation to the ideal filter. It is a piecewise continuous and circulary symmetric filter given by

$$B(K_x, K_y) = \frac{1}{1 + \frac{(K)^{2n}}{\sqrt{K_x^2 + K_y^2}}}$$

The parameter n is a user-defined positive integer called the order of the filter.

Note: As the value of n increases, the BHPF approaches the ideal filter.

3.11.3.3. The Exponential Highpass Filter

The exponential highpass filter (EHPF) with cut-off frequency at a distance D_0 from the original has a transfer function given by

$$E(K_x, K_y) = \frac{1}{\exp\left[\dfrac{D_0}{\sqrt{K_x^2 + K_y^2}}\right]^n}$$

where n controls the rate of increase of E(u,v)

Note: when $\sqrt{u^2 + v^2} = D_0$ then E(u,v) =1/e, that needs some modification to be equal $\frac{1}{\sqrt{2}}$ of its maximum value at the cut-off frequencies.

3.11.3.4. The Trapezoidal Highpass Filter

A trapezoidal highpass filter (THPF) can be defined by

$$T(K_x, K_y) = \begin{cases} 0 & , \quad \text{if } \sqrt{K_x^2 + K_y^2} < D_1 \\ \dfrac{1}{[D_0 - D_1]} \left[\sqrt{K_x^2 + K_y^2} - D_1\right] & , \text{ if } D_1 \le \sqrt{K_x^2 + K_y^2} \le D_0 \\ 1 & , \quad \text{if } \sqrt{K_x^2 + K_y^2} > D_0 \end{cases}$$

where D_0 and D_1 are specified and it is assumed that $D_0 > D_1$.

3.11.4. The Homomorphic Filtering

Homomorphic filtering assumes an **illumination-reflection** model for the image of the form

$$f(x,y) = i(x,y)\, r(x,y)$$

where r is the reflection component, is the illumination and f is the intensity distribution. In general, this model has two important features which homomorphic filtering relies upon. They are:

1. The illumination component is composed of low spatial frequencies.
2. The reflection component is composed of high spatial frequencies.

The objective of the homomorphic filtering is to separate the reflection component from the illumination component.

The reflection component in homomorphic filtering provided by the equation

$$\mathbf{r(x,y) \approx \exp\left(HPF\left[\ln f(x,y)\right]\right)}$$

This method of image enhancement is known as homomorphic filtering. It requires specification of the HPF. A variety of filters HPF's can be used for this purpose. Image enhancement by homomorphic filtering is usually carried out using the BHPF. In practice, a 2D FFT is used to carry out the filtering operation and the user may repeat this enhancing process for different values of k and n.

3.11.5. The Laplacian Filter

The Laplacian of two-variable function f(x,y) is defined as

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

It follows from the definition of the two-dimensional Fourier transform that

$$\Im\left\{\nabla^2 f(x,y) \Leftrightarrow -(2\pi)^2\left(K_x^2 + K_y^2\right)F(K_x,K_y)\right.$$

The Laplacian operator (filter) is useful for outlining edges in an image.

3.11.6. High Emphasis Filtering

High emphasis filtering is based on computing the Laplacian of a blurred

image and then subtracting the result from the image. This process provides a reconstruction which is particularly good at restoring the edges of the blurred image. The high emphasis filtered image (which shall be denoted by g) is related to the blurred image f by the formula

$$g(x,y) = f(x,y) - \nabla^2 f(x,y)$$

The effect of the Laplacian is to amplify the high spatial frequencies in the image while leaving the low frequencies relatively unchanged.

3.11.6.1. Physical model for High Emphasis Filtering

High emphasis filtering is based on a physical model for the image which assumes that the blurring has occurred via a process of diffusion. This process is described by the following partial differential equation (the diffusion equation).

$$\frac{\partial}{\partial t} f(x,y,t) = K\nabla^2 f(x,y,t)$$

where $K > 0$ is a constant which determines the rate of diffusion and

$$f(x,y,t) = g(x,y), \; t = 0$$

At any time $t > 0$, it is assumed that a diffusion process is responsible for blurring the image g.

Note: As time increases, the image becomes progressively blurred.

The problem is to find g from f at some $t > 0$. Suppose that we record the diffusion blurred image f at a time $t = T$. In this case the solution for g is

$$g(x,y,0) = f(x,y,T) - TK\nabla^2 f(x,y,T)$$

The equation for high emphasis filtering given earlier applies to the case when TK = 1 but the value can be changed if the user requires some control over the process. Introducing the scaling factor α we can use the equation (t = T).

$$g(x,y) = f(x,y) - a\nabla^2 f(x,y,t)$$

The effect of increasing the value α is to amplify the high frequency content of f. By lowering the value of α the influence of the high frequencies in f or g is decreased.

3.11.6.2. Computation of the High Emphasis Filter

Image enhancement by high emphasis filtering is usually achieved by computing the digital Laplacian. This is given by

$$\nabla^2 f_{i,j} = f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} - 4f_{i,j}$$

from this result, we have

$$g_{i,j} = f_{i,+1,j} - \nabla^2 f_{i,j}$$
$$5f_{i,j} - f_{i+1,j} - f_{i-1,j} - f_{i,j+1} - f_{i,j-1}$$

3.11.6.3. The High Emphasis Filter as a Convolution Kernel

The digital Laplacian is a shift invariant linear operation. Applying the operation to a digital image f_{ij} is the same as convolving the image with the two-dimensional array.

$$\begin{matrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{matrix}$$

where all other points in the array are zero. Computing g_{ij} is the same as convolving f_{ij} with the array

$$\begin{matrix} & -1 & \\ -1 & 5 & -1 \\ & -1 & \end{matrix}$$

We can write

$$\mathbf{g_{ij} = Kernel_{ij} \otimes\otimes f(i,j)}$$

where

$$\mathbf{Kernel_{ij} = \begin{matrix} & -1 & \\ -1 & 5 & -1 \\ & -1 & \end{matrix}}$$

If the scaling parameter α is included, then the Kernel above becomes

$$\mathbf{Kernel_{ij} = \begin{matrix} & -a & \\ -a & 4a+1 & -a \\ & -a & \end{matrix}}$$

The convolution of an image with the Kernels of this size can be completed efficiently using FIR filter.

3.12. Noise Reduction

Noise refers to a variety of unwanted disturbances due to measuring and recording errors of all types and interference from external recourses. Depending on the type of experiment that is concerned, noise may be confined to a range of frequencies or exist over the entire spectrum. Noise signals which have a broad spectrum are referred to as **White Noise**. Noise signals which have a broad spectrum confined to a particular (narrow) band are referred to as coloured noise. In many cases noise in an image is primarily a high frequency effect and corrupts data where the energy of the data spectrum is low.

One way of reducing noise is by attenuating the high frequency components of the data over a range of frequencies which can be selected and adjusted by the user to provide optimum results. This can be achieved by applying a low pass filter.

3.12.1. The Lowpass Filter

As the name suggests, the low pass filter passes the low frequencies of the data spectrum and attenuates the high frequencies.

3.12.1.1. The Ideal Lowpass Filter

The ideal lowpass filter is given by

$$H(K_x, K_y) = \begin{cases} 1, & \sqrt{K_x^2 + K_y^2} \le K \\ 0, & \sqrt{K_x^2 + K_y^2} > K \end{cases}$$

where K is the cut-off frequency. This type of filter attenuates completely all the spatial frequencies above the cut-off frequency and retains without modification all those frequencies less than or equal to the cut-off frequency.

3.12.1.2. The Gaussian Lowpass Filter

The discontinuous nature of the ideal lowpass filter causes ringing to occur in the filtered output. This phenomenon is known as the Gibbs effect. The Gibbs effect can be reduced by employing lowpass filters which are piecewise continuous functions in the spectral domain. A number of filters are available for this purpose such as the Gaussian lowpass filter given by

$$G(K_x, K_y) = \exp\left[-\left(K_x^2 + K_y^2\right) \Big/ 2\sigma^2 \right]$$

where σ^2 is the width of the filter at 1/e. By adjusting the value σ the bandwidth of this filter can be adjusted.

3.12.1.3. The Butterworth Lowpass Filter

Another widely used filter of this type is the Butterworth Lowpass filter (BLPF) which is given by

$$B(K_x, K_y) = \frac{1}{1 + \left(\frac{\sqrt{K_x^2 + K_y^2}}{K} \right)^{2n}}$$

In this expression, n is referred to as the order of the filter. This parameter is a positive integer and determines the rate at which the filter approaches zero. The Butterworth filter is a good approximation to the ideal lowpass filter which avoids ringing in the output.

3.12.1.4. The Trapezoidal Lowpass Filter

A trapezoidal lowpass filter (TLPF) is a compromise between the TLPF and a completely smooth filter. The TLPF is defined by

$$T(K_x, K_y) = \begin{cases} 1 & , \quad \text{if } \sqrt{K_x^2 - K_y^2} < D_0 \\ \frac{1}{[D_0 - D_1]} \left[\sqrt{K_x^2 - K_y^2} - D_1 \right] & , \text{ if } D_0 \le \sqrt{K_x^2 - K_y^2} \le D_1 \\ 0 & , \quad \text{if } \sqrt{K_x^2 - K_y^2} > D_0 \end{cases}$$

where D_0 and D_1 are specified and it is assumed that $D_0 < D_1$.

3.12.2. The Neighbourhood Averaging (Mean) Filter

The neighbourhood averaging filter is a spatial domain technique. A window is chosen which encloses a pre-determined neighbourhood of pixels. The average value of the pixels enclosed by this window is then computed and usually assigned to the pixel at the centre of the neighbourhood. By moving the window and repeating this process, a neighbourhood averaged image is obtained. The size of neighbourhood is defined by the user. The effect of computing the average of a neighbourhood of pixels is to eliminate any sudden jumps in the grey level which could be caused by some noise process. In mathematical terms, we can express the neighbourhood averaging process as

$$g(i,j) = \frac{1}{M} \sum_{(n,m) \in S} f(n,m)$$

where S is the window enclosing n x m neighbours whose centre is located at (i,j) and M is the total number of pixels (enclosed by S).

3.12.3. The Median Filter

The aim of all noise-reducing processes is to suppress noise without blurring or degrading the original image. The neighbourhood averaging filter does not preserve the sharpness of an image because the averaging process smoothes the data. The median filter is superior to the moving average filter in that it is better at preserving sharp features while eliminating noise, especially isolated noise spikes. The basic data is the same as the neighbourhood averaging filter except that instead of computing the average of the neighbourhood we compute the median of the neighbourhood.

3.13. Segmentation

Segmentation is concerned with the process of dividing an image into meaningful regions or segments. It is used in image analysis to separate or isolate features from the background.

Segmentation is based on one of two properties in an image:

i) Similarity

ii) Discontinuity

The first property segments an image into regions which have grey level within a pre-determined range. The second property segments an image into regions of discontinuity where there is a more or less abrupt change in values of the grey levels. This is used to detect the boundaries or edges of features in the image and is consequently known as edge detection.

3.13.1. Thresholding

Thresholding is a relatively simple approach to segmenting an image into regions of similarity.

Basic idea: group pixels within a common range of grey levels into a pre-determined set.

3.13.1.1. Single Band Thresholding

Single band thresholding converts an image of this type into binary form consisting of just 0's and 1's by application of the following processes:

If $\mathbf{u_{in}(i,j) > threshold}$

$\mathbf{u_{out}(i,j) = 1}$

Else

$\mathbf{u_{out}(i,j) = 0}$

Endif

where 0 < threshold < 1

3.13.1.2. Semi-thresholding

In some cases, it is preferable to retain the grey level variations that occur above the threshold. This is known as semi-thresholding and is achieved by applying the following process:

If $\mathbf{u_{in}(i,j) > threshold}$

$$\mathbf{u_{out}(i,j) = u_{in}(i,j)}$$

Else

$$\mathbf{u_{out}(i,j) = 0}$$

Endif

3.13.1.3. Multiband fixed thresholding

This technique is based on the following process:

$$\mathbf{u_{out}(i,j) = 0 \text{ If } 0 \le u_{out}(i,j) \le t_1}$$
$$\mathbf{u_{out}(i,j) = 1 \text{ If } t_1 \le u_{in}(i,j) \le t_2}$$
$$\mathbf{u_{out}(i,j) = 2 \text{ If } t_2 \le u_{in}(i,j) \le t_3}$$

and so on where t_1, t_2, t_3,... are different thresholds.

3.13.2. Edge detection

Edge detection is basically a method of segmenting an image into regions of discontinuity i.e. it allows the user to observe those features of an image where there is a more or less abrupt change in grey level or texture - indicating the end of one region in the image and the beginning of another. Edge detection makes use of differential operators to detect changes in the gradients of the grey levels.

Edge detection is divided into two main categories:

i) First order edge detection

ii) Second order edge detection

As the name suggests, first order edge detection is based on the use of first derivatives whereas second order edge detection is based on the use of second derivatives, in particular, the Laplacian ∇^2.

3.13.2.1. First order edge detection

First order edge detection is based on computing the gradient of an image at (x,y) and observing the locations in the image where it changes abruptly.

If the image is I(x,y), then the basic idea is to compute

$$\nabla \mathbf{I}(\mathbf{x},\mathbf{y}) = \hat{\mathbf{x}}\frac{\partial}{\partial \mathbf{x}}\mathbf{I}(\mathbf{x},\mathbf{y}) + \hat{\mathbf{y}}\frac{\partial}{\partial \mathbf{y}}\mathbf{I}(\mathbf{x},\mathbf{y})$$

3.13.2.2. Digital gradients

Different gradients methods used for edge detection result from attempts to find digital approximations to ∇I. The approximations available are compounded in a class of operators known as digital gradients. The equivalence of digital gradient operations and discrete convolutions with certain types of kernels is an important result in the theory and practice of edge detection.

Robert's gradient

$$\mathbf{G}_{ij} = \sqrt{\left[\mathbf{I}_{ij} - \mathbf{I}_{i+1,j+1}\right]^2 + \left[\mathbf{I}_{i+1,j} - \mathbf{I}_{i,j+1}\right]^2}$$

The Robert's gradient can be calculated using a FIR filter with the kernels

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

3.13.2.3. Other first order edge detectors

- **Sobel operator**

Convolution kernels are:

x-direction

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

y-direction

$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

- **Prewitt operator**

Convolution kernels are:

x-direction

$$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

y-direction

$$\begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

- **Orthogonal gradients**

Orthogonal edge detection uses three directions x direction, y direction and cross direction. This can be seen in the equation below:

$$G_{ij} = \sqrt{\begin{aligned}&\left[I_{ij} - I_{i,j+1}\right]^2 + \left[I_{i+1,j} - I_{i+1,j+1}\right]^2 + \left[I_{ij} - I_{i+1,j}\right]^2 \\ &+ \left[I_{i+1,j} - I_{i+1,j+1}\right]^2 + \left[I_{ij} - I_{i+1,j+1}\right]^2 + \left[I_{i+1,j} - I_{i,j+1}\right]^2\end{aligned}}$$

3.13.2.4. Second order edge detection

Second order edge detection is based on computing the second derivative in x and y and observing the locations in the image where zeros occur.

The Laplacian of an image given by

$$\nabla^2 I(x,y) = \frac{\partial^2}{\partial x^2} I(x,y) + \frac{\partial^2}{\partial y^2} I(x,y)$$

is used for this purpose. The positions where $\nabla^2 I$ changes from + to - or from - to + is then computed and displayed. The digital Laplacian operating on an image $I(i,j)$ becomes

$$\nabla^2 I_{i,j} = I_{i+1,j} + I_{i-1,j} + I_{i,j+1} + I_{i,j-1} - 4I_{i,j}$$

The operation can be written in terms of the convolution of I with an appropriate kernel, i.e.

$$\nabla^2 I_{ij} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \otimes\otimes I_{ij}$$

3.13.2.5. Edge enhancement using Artificial Neural Networks

One key problem in digital images is how to detect specific features in an image. It turns out that even finding a simple feature such as an *edge* can be difficult.

One way to solve the edge enhancement problem is by using Artificial Neural Networks (ANN). Although the techniques discussed here were suggested by the processes of the human eye, they are not intended to be biologically accurate, nor is the solution intended to be biologically plausible.

3.13.2.6. A logical Artificial Neural Networks Model

Consider the surface of the eye as an array or grid of photoreceptive elements. Light from the outside world impinges on this photoreceptive array and provokes output from each of the array elements. The output of each of these photoreceptors is passed on to another layer of corresponding neurones that work together to enhance the image.

The layers are excites shown by "+" and the other layers are inhibits shown by "-".

```
-●  -●  -●  -●  -●
-●  +O  +O  +O  -●
-●  +O  +O  +O  -●
-●  +O  +O  +O  -●
-●  -●  -●  -●  -●
```

First, the ANN computes the Internal activation which is given by

$$\mathbf{I} = \sum \mathbf{W}_j \mathbf{X}_j - \mathbf{Bias}$$

where

I is the Internal activation,

W is the Weights,

X is the output layer (element) and

Bias is the weight Bias-to-output layer.

To produce an actual output (Target layer) the internal activation value I is transformed by the non-linear transfer function F.

$$\mathbf{F(I)} = \begin{cases} \mathbf{0}, & \mathbf{I \leq 0} \\ \mathbf{1} & \mathbf{otherwise} \end{cases}$$

Note: This type of Artificial Neural Networks is fed back onto itself F as part of the input for computing I (see Fig. 3.1).

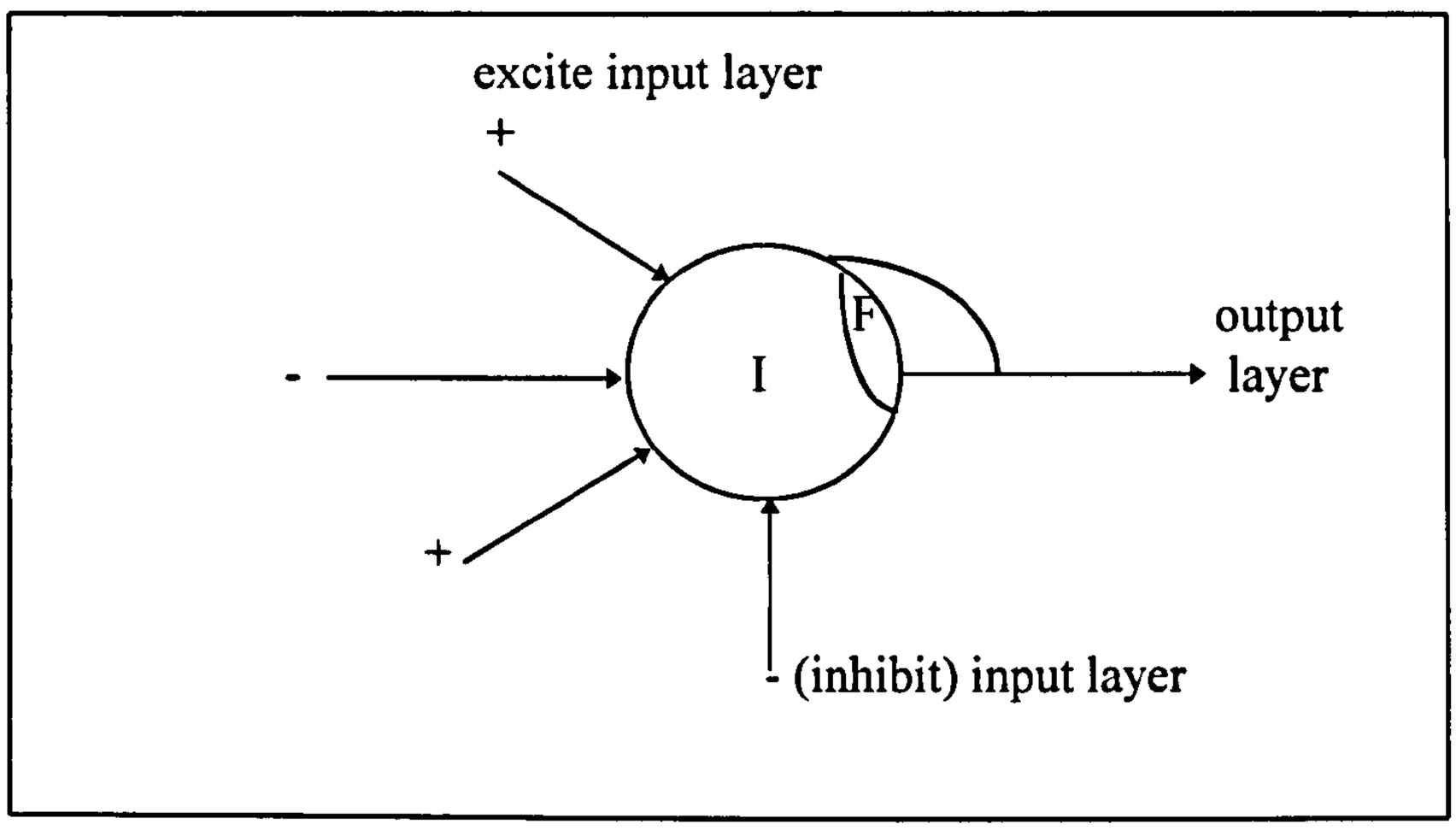


Fig 3.1 Artificial Neural Networks

3.14. Computer Aided Tomography (CAT) - Reconstruction from Projections

CAT is used in a wide range of applications, most notably for medical imaging.

The mathematical basis of this mode of imaging is compounded in an integral transform called the **Radon** transform. There are a number of different techniques which may be used to compute the Radon transform and its associate inverse. These techniques include:

i) back-projection and deconvolution

ii) filtered back-projection and

iii) the central slice theorem

3.14.1. Background to Computer Aided Tomography

Computer aided tomography (CAT) can be considered to be a special type of image reconstruction problem. This problem is concerned with evaluating the structure of an object from observations on how it modifies certain properties of a probe. CAT is usually concerned with projection tomography - the reconstruction of an object from a sequence of projections. This problem is compounded in the inversion of an integral transform called the **Radon** transform which was derived by the Austro-German mathematician Johannes Radon. Radon proved that **the complete set of one-dimensional projection obtained from a continuous two-dimensional function, contains all the information required to reconstruct this same function.**

What is a projection?

A projection is obtained by integrating a function over a set of parallel lines and is characteristic of its angle of rotation in the cartesian plane.

Parallel lines	Straight-Line projection
Divergent lines	fan-bean projection

By rotating the function in the cartesian plane, a different projection is obtained. The reconstruction problem: Given a set of projections obtained by rotating a 2D function through different angles, reconstruct the function from which the projections have been obtained. The solution to this problem is given by the inverse Radon transform.

3.14.2. The Radon transform

Consider an inhomogeneous object of compact support, defined in a 2D cartesian space by the object function O(x,y). The mapping defined by the projection or line integral of O along all possible line L can be written in the form

$$\mathbf{P} = \int_{\mathbf{L}} \mathbf{O}(\mathbf{x},\mathbf{y})\,\mathbf{dL}$$

where dL is an increment of length along L. This equation represents a mapping from (x,y) cartesian co-ordinates to (Z,ϑ) polar co-ordinates. This is indicated explicitly by it in the form

$$\mathbf{P}(\mathbf{Z},\vartheta) = \int_{\mathbf{L}(\mathbf{Z},\vartheta)} \mathbf{O}(\mathbf{x},\mathbf{y})\,\mathbf{dL}$$

If $P(Z,\vartheta)$ is known for all Z and ϑ, then $P(Z,\vartheta)$ is the Radon transform of O(x,y). This is indicated by writing

$$\mathbf{P} = \hat{\mathbf{R}}\,\mathbf{O}$$

where $\hat{R}$ is the Radon transform operator.

Using 2D vector notation an alternative but equivalent of writing P is

$$\mathbf{P}(\mathbf{Z},\vartheta) = \int \mathbf{O}\left(\vec{\mathbf{r}}\right)\mathbf{d}\left(\mathbf{Z}-\vec{\eta}.\vec{\mathbf{r}}\right)\mathbf{d}^{2}\vec{\mathbf{r}} = \hat{\mathbf{R}}\,\mathbf{O}\left(\vec{\mathbf{r}}\right)$$

3.14.3. Reconstruction methods

The formula for reconstructing a function from its Radon transform is given by

$$\mathbf{O}\left(\hat{\mathbf{R}}^{-1}\right)\mathbf{P}\left(\hat{\eta},\mathbf{Z}\right) = -\frac{1}{2\pi^{2}}\int_{0}^{\mathbf{p}} \mathbf{d}\vartheta \int \mathbf{dZ}\,\frac{1}{\mathbf{Z}-\hat{\eta}.\vec{\mathbf{r}}}\,\frac{\partial}{\partial \mathbf{Z}}\mathbf{P}\left(\hat{\eta},\mathbf{Z}\right)$$

where R^{-1} is the inverse Radon transform operator. This formula is always valid in cases where P is continuous over an infinite set of projections for all

lines, rather than a discrete set. Thus a digital reconstruction process based on the above equation will therefore only be an approximation of the actual object by non-unique approximations.

3.14.3.1. Back projection and deconvolution

Given the data $P(Z,\vartheta)$, attempt to reconstruct $O(x,y)$ by back-projecting and then **adding** the results. Back-projection provides a reconstruction of the form

$$B(x,y) = P(x,y) \otimes\otimes O(x,y)$$

B-Back-Projection Function

O-Object Function

P-Point Spread Function

To compute O, we must find P and then deconvolve B. It can be proved that

$$P(x,y) = \frac{1}{\sqrt{x^2 + y^2}}$$

The fourier transform of P is given by

$$P(K_x, K_y) = \frac{1}{\sqrt{K_x^2 + K_y^2}}$$

The back projection function is given by

$$B(x,y) = \frac{1}{\sqrt{x^2 + y^2}} \otimes\otimes O(x,y)$$

Taking the 2D Fourier transform of both sides and using the convolution theorem

$$\tilde{B}(x,y) = \frac{1}{\sqrt{K_x^2 + K_y^2}} \tilde{O}(K_x, K_y)$$

Rearranging, we have

$$O(x,y) = \hat{F}_2^{-1}\left[\sqrt{K_x^2 + K_y^2}\, \hat{B}(K_x, K_y)\right]$$

Note: $\sqrt{K_x^2 + K_y^2}$ is a **non-singular** inverse filter.

Hence, for back-projection and deconvolution we design an algorithm based on

$$\mathbf{O(x,y) = \hat{F}_2^{-1}\left[\left|\vec{K}\right| \hat{F}_2 \hat{B}\right]}$$

3.14.3.2. Filtered Back-Projection

We can analyse the inverse Radon transform on terms of its constituent operators.

Analytical form	Name	Operator
$\frac{\partial}{\partial Z}$	Differentiation	∂z
$\frac{1}{\pi}\int_{-\infty}^{\infty}\frac{dz}{z-\hat{\eta}.\vec{r}}$	Hilbert transform	$\hat{H}$
$-\frac{1}{2\pi}\int_{0}^{p} dv$	Back-projection	$\hat{B}$

Using operators defined above we can write

$$\mathbf{O(x,y) = \hat{B}\hat{H}\partial_z P(z,v)}$$

To compute the Inverse Radon Transform:

i) Differentiate

ii) Hilbert transform

iii) Back-projection

$\hat{H}\,\partial_z$ - Filter operator

$\hat{B}$ - Back-projection operator

Hence the name **Filtered Back-Projection**. The form of the filter is obtained by computing the filter corresponding to the real space operation $\hat{\mathbf{H}}\partial_z$

Real Space X	**Fourier Space** K
∂z	**iK**
$\hat{\mathbf{H}}$	**-i sgn(k)**
$\hat{\mathbf{H}}\partial_z$	**K sgn(K)=$\lvert K\rvert$**

Hence, for filtered back-projection we design an algorithm based on

$$\mathbf{O(x,y)=\hat{B}\,\hat{F}_1^{-1}\left[|K|\,\hat{F}_1\,P(Z,v)\right]}$$

3.14.3.3. Reconstruction via the Central Slice Theorem

The Central Slice Theorem provides a relationship between the Radon transform of an object and its two dimensional Fourier transform. The theorem shows that **the one dimensional Fourier transform of a projection at a given angle θ is equal to the function obtained by taking a radial slice through the two dimensional Fourier domain of the object at the same angle θ.**

Using operator rotation we can write this statement in the form

$$\mathbf{\hat{F}_2\,O(x,y)=\hat{F}_1\,\hat{R}\,O(x,y)}$$

Reconstruction via the Central Slice Theorem is based on algorithms which utilise the relationship

$$O(x,y) = \overset{n}{F}_{2}^{-1}\left[\hat{F}_1 R \hat{O}\left(\vec{r}\right)\right]$$

Note: All the implementation of the results of this chapter are given in Appendix **D**.

References

Blackledge J.M, (1993/94), *C Programming*, MSc Lecture Notes, De Montfort University, School of Mathematical and Computing Sciences.

Blackledge J.M, (1993/94), *Digital Image Processing*, MSc Lecture Notes, De Montfort University, School of Mathematical and Computing Sciences.

Lindley C.A, (1991), *Practical Image Processing in C*, John Wiley & Sons, Inc, USA.

Rafael C Gonzalez, Paul Wintz, *Digital Image Processing*, 1977.

William K Pratt, *Digital Image Processing*, 1978.

Introductory Computer Vision and Image Processing by **Adrian Law**, 1991.

Chapter 4

Artificial Neural Networks and Their Applications

This chapter explains in great detail what Artificial Neural Networks (ANN) model is and its Back-Propagation Algorithm.

In this chapter **Page No**

4.1. Introduction

4.1.1. Artificial Neural Networks

Artificial Neural Networks (ANNs) is a powerful and practical tool for solving problems that would be difficult using conventional computer science techniques. The progress in the adaptation of Neural Networks as a problem-solving tool in the last few years is widespread, it has been proved successfully on a wide range of problems for object and pattern recognition (image recognition, speech recognition, etc.).

The three-layer neural network model has been widely implemented since it was discovered in 1980's and it is now one of the best known Neural Network model [1],[2]. This three-layer ANN model has one input-layer with three nodes, one hidden-layer with one node and one-output-layer with three nodes.

In this research we use fully connected three-layer ANN model with one input-layer with 64 nodes, one hidden-layer with 64 nodes and one-output-layer with 64 nodes, Back-Propagation algorithm network (**Fig 4.2**).

In our model we used fully connected 64-nodes to represent an image block of size 8x8 which is standard sized to avoid two major problems **(i)** memory space problem **(ii)** running time (time consumption) problem. Therefore, all the blocks are of size 8x8.

The neural network model was trained using the data compression method with a network on the un-preprocessed data image (**see section 4.4.**). The result of these experiments indicates that neural networks are capable of learning to recognise character images that have been transformed with similar levels of accuracy to network trained on un-preprocessed images (**see section 4.4.2.**) for more explanation about un-preprocessing images and pre-

processing images. BP is the most widely applied and successful algorithm for the training stage (**see section 4.2.**). However, there is one problem related to the use of the BP algorithm and that is the time required to converge to small errors needs a large number of patterns, especially for large networks and complex patterns [3],[4],[5],[7].

In this research we have attempted to solve the above problems by using a Fractal Image Compression algorithm to speed up the process by compressing a large image to a small one and prepare it for the ANN model.

4.1.2. ANNs and the C Programming Language

The neural network model and its software are coded in the **C** language instead of FORTRAN language which is traditionally one of the most popular language in image processing applications. This is because **C** is a general purpose programming language which features economy of expression, modern control flow and data structures (especially memory access) and has pointers which is the best technique to compact and efficiently a major problem in image processing applications. Neural network models run on a large amount of memory space and the FORTRAN language does not have pointers to deal with such a memory space problem (FORTRAN-90 has, but this is not implemented in this research). The **C** language was originally designed for and implemented on the UNIX operating system and essentially all UNIX applications are written in **C** which helps the user create a library in a very simple and easy way. Also the **C** compiler is much better at handling error messages than the FORTRAN compiler. The C language has proven to be a pleasant, expressive and versatile language for a wide variety of programs [5],[6].

4.2. The Back Propagation Algorithm

4.2.1. Introduction

Learning problems occur in ANNs that are required to map a well defined set of input units into a well defined set of output units. They can generally be solved by the introduction of hidden units. The **BP** algorithm is the prescription originally suggested by *Rumehart Hinton and Williams* (1986) for dealing with the training of these hidden units [12]. The physical processes by a *feedforward* and *feedbackward* neural network architecture using the BP algorithm when successfully trained, maps the input parameters to the desired output. The neural network can learn in two different modes:

1. The incremental mode:

In this mode the updating occurs after each pattern is presented i.e. update the input unit by multiplying the updating weight with output units (**see section 4.3.3.**).

2. The batch mode:

In this mode, updates increments are accumulated and updating takes place after all patterns have been presented (**see section 4.3.3.**).

The aim of the BP algorithm is to update the coupling weights in a neural network, i.e. to minimize the squared differences between the desired output (target) and the net output (**Fig 4.1**) shows the minimization function, where the w_{12} is the weight between the output and hidden layer, Δw_{12} is the new net output and $f_j(a)$ is the activation function.

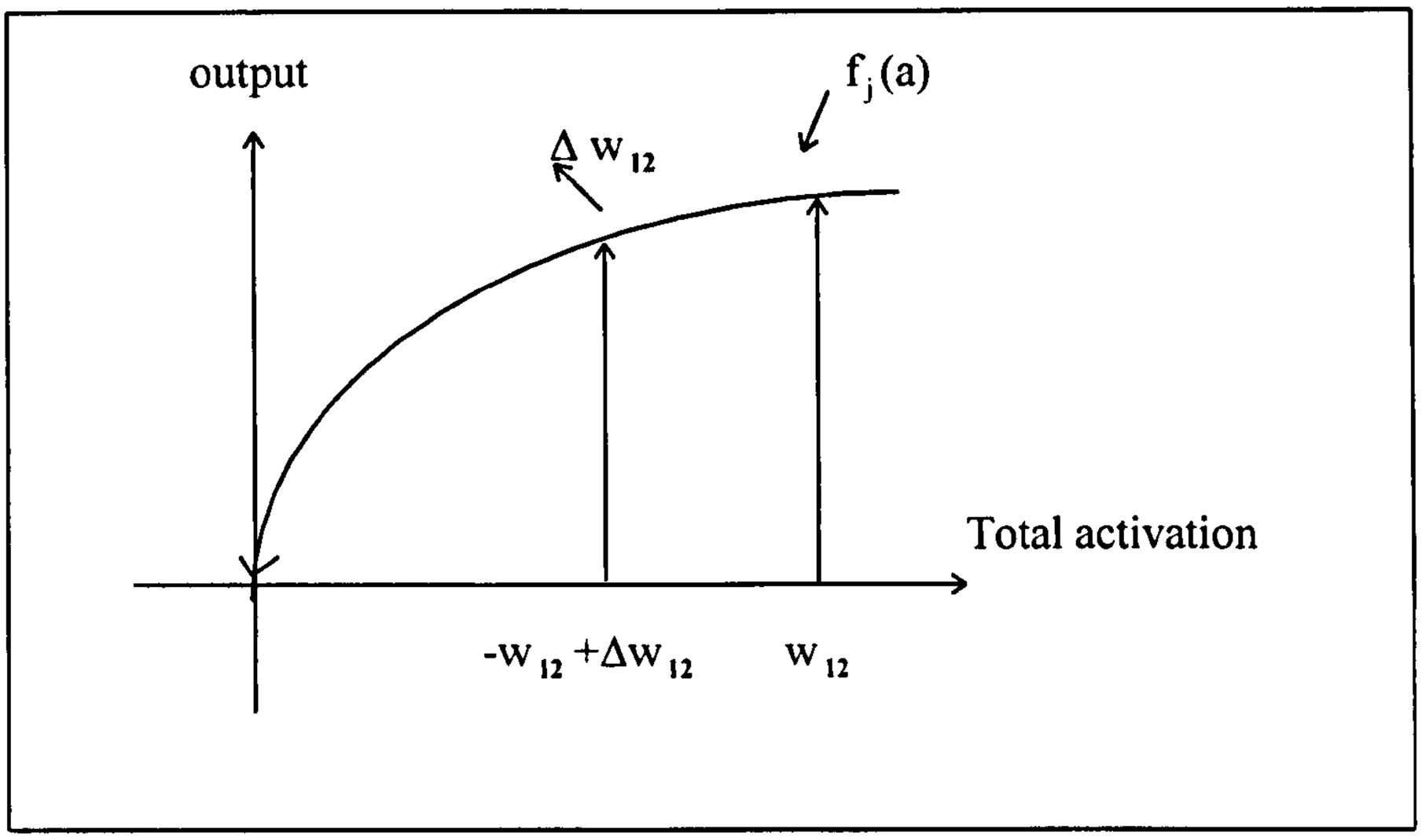


Fig 4.1. The Minimization Function

4.2.2. Principles of the Back-Propagation (BP) Algorithm

4.2.2.1. Introduction

The **BP** is constantly being revised and improved, so the contents of this chapter merely provide an introduction to the fundamental version of both the theory and mathematical phase of the algorithm. The **BP** model is a steepest descent algorithm in which each weight in the network is adjusted to minimize its contribution to the total mean square error between the desired (target) and actual system outputs.

4.2.2.2. The Layout of the BP Algorithm

The **BP** model is usually described as a three level model, with level **F1, F2** and **F3**, such that level **F2** is a level of "hidden units" between **F1** and **F3**, the purpose of the model is to learn an associative map [13] between the input level **F1** and the output level **F3**. The map is designed to sufficiently distributed and allow alterations in the inputs at **F1** to generate appropriate

alterations in the outputs at **F3**. Such a possibility depends upon general projection properties of distributed associative maps (Kohonen, 1984). The key property demonstrated by computer simulations of the **BP** model is that it can learn a distributed associative map. However, it needs an external teacher **(see section 4.3.2.)**. In addition to the levels **F1, F2, F3** and the pathways **F1→F2→F3,** the **BP** model also requires levels **F4, F5, F6 and F7** as well as a complicated set of highly specific interactions between these levels and the rest of the network **(Fig 4.2 and Fig 4.3)**. These levels and interactions will now be described in **(section 4.2.3.)**.

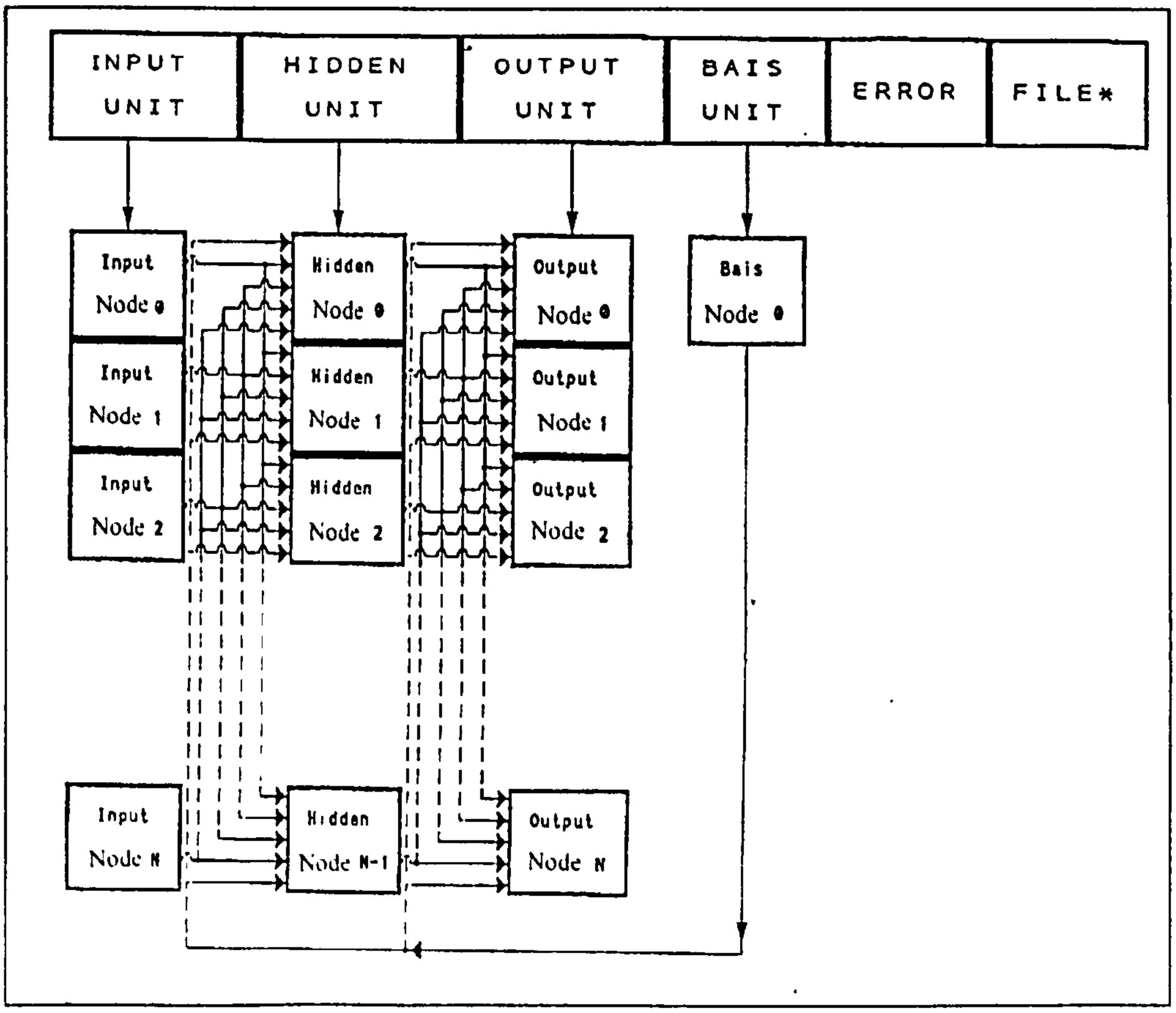


Fig 4.2 The Artificial Neural Networks Model

4.2.2.3. The Flow Paths and Process Directions

Inputs delivered to **F1** propagate forward through **F2** to **F3**, where they generate the actual outputs of the network. The desired, or expected outputs are independently delivered to level **F4** by an external teacher on every learning trial (**see section 4.3.2.**). The actual outputs at **F4** generate error signals. These error signals propagate from **F4** to the **F2→F3** pathways, where they change the weights in the **F2→F3** pathways.

4.2.2.4. The Technique of the BP Algorithm

The **BP** proceeds as follows:

The weights computed in the top-bottom **F2→F3** pathways are transported to the bottom-up **F4→F5** pathways. Once in these pathways, the differences between expected and real outputs at **F4** are multiplied by the transported weights within the **F4→F5** pathways to generate weighted error signals which change the weights in the **F1→F2** pathways. Such a physical transport of weights has no plausible physical interpretation. The weights in the **F2→F3** pathways must be computed *within* these pathways in order to multiply signals from **F2 to F3 (Fig 4.3)**. These weights cannot exist *within* the pathways from **F4** to **F5** in order to multiply signals from **F4** to **F5** *without* being physically transported from (**F2→F3**) to (**F4→F5**) pathways, thereby violating basic properties of locality. Moreover, the levels **F3** and **F4** can not be lumped together, because **F3** must record actual outputs, whereas **F4** must record differences between expected and actual outputs.

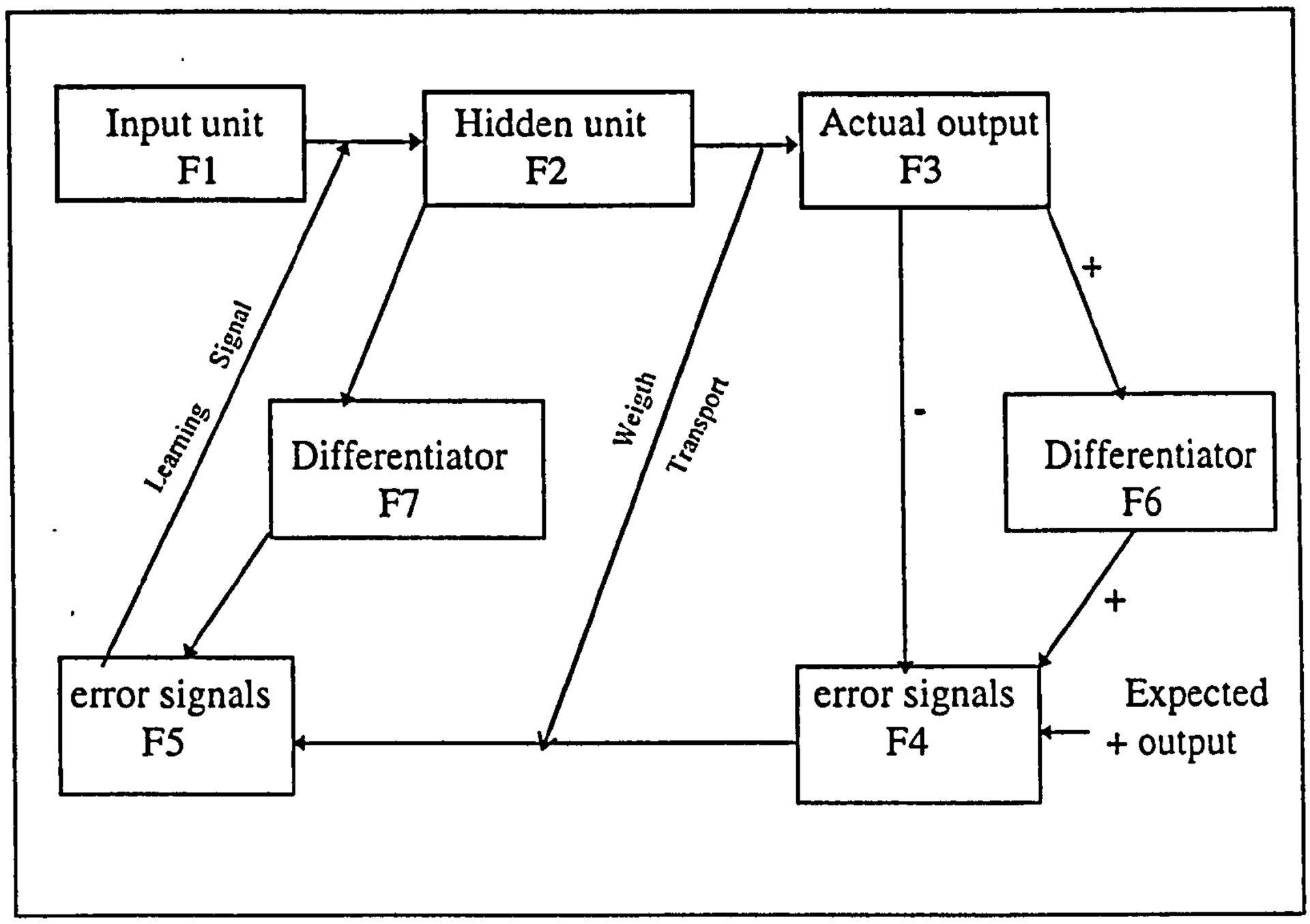


Fig 4.3 Circuit diagram of BP model

4.2.2.5. Correction of the System Output

The computation of the error signal has an additional **complexity** (see **section 4.3.4. (equation 2.4) and section 4.3.5 (equation 2.5)**). In addition to subtracting each actual output at **F2** from each expected output at **F4**, the derivative of each actual output is also computed. The difference between each expected and actual output is multiplied by the corresponding derivative (see **section 4.3**), in addition to being multiplied by the corresponding transported weight. Thus there exists additional levels **F6** and **F7** at each layer for converting and signalling these derivatives, with great position specificity to the correct transported weights (**Fig 4.1**). This complex interaction scheme must be replicated at every stage of hidden units that is used in the BP model [10] (**Fig 4.2**).

4.3. The Differential Equations of the BP Algorithm

Consider a network model with two layers i.e. consisting of one hidden layer and an output layer (**Fig 4.4**) which shows a two layer network consisting of three units).

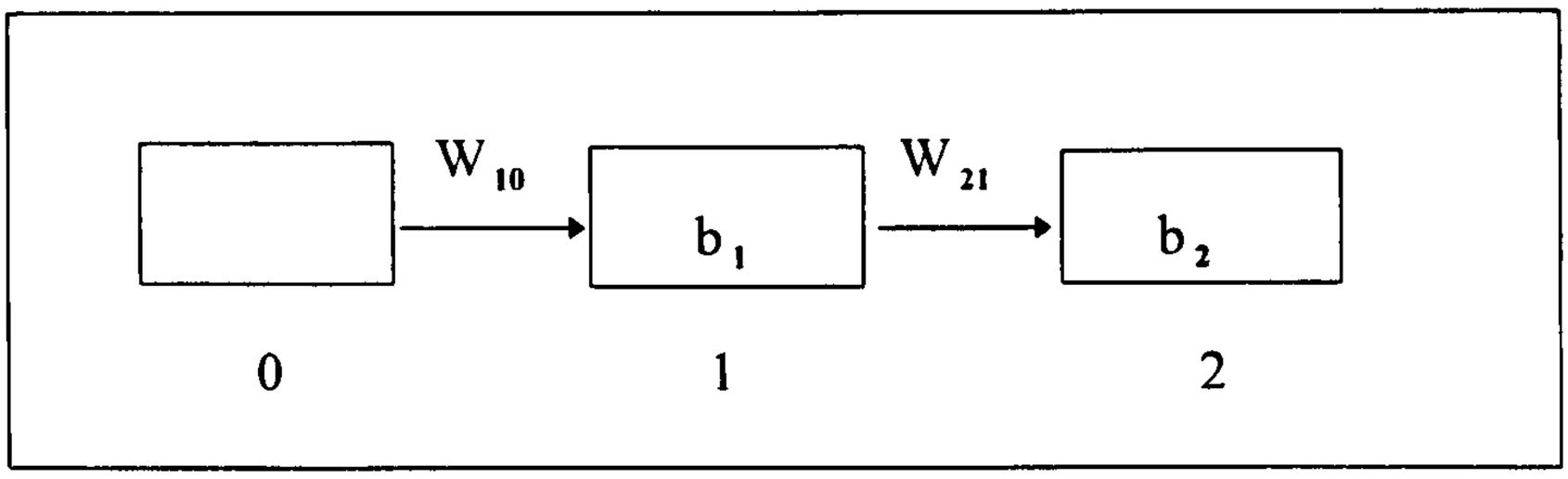


Fig 4.4 Two Layers Network

4.3.1. The Neural Network Equations

The system dynamics in the search phase of a two layer Neural Network with three units for a single input/output pair (**Fig 4.4**) can be described by the following equations:

$$\mathbf{h}_1 = \mathbf{w}_{10}\mathbf{i}_0 + \mathbf{b}_1$$

$$\mathbf{v}_1 = \mathbf{g}(\mathbf{h}_1)$$

$$\mathbf{h}_2 = \mathbf{w}_{12}\mathbf{v}_1 + \mathbf{b}_2$$

$$\mathbf{v}_2 = \mathbf{g}(\mathbf{h}_2)$$

where $\mathbf{h}_1$, $\mathbf{w}_{10}$ and $\mathbf{b}_1$ are the parameters used in the hidden layer,

$\mathbf{h}_2$, $\mathbf{w}_{21}$ and $\mathbf{b}_2$ are the parameters used in the output layer,

$\mathbf{g}$ is the non-linear activation function,

$\mathbf{v}_1$ is the activation value of the unit in the hidden layer,

$\mathbf{v}_2$ is the activation value of the unit in the output layer,

$\mathbf{i}_0$ is the input layer,

$\mathbf{b}_1$ is the bias of the unit in the hidden layer,

$\mathbf{b}_2$ is the bias of the unit in the output layer,

$\mathbf{w}_{10}$ is the weight between the hidden layer and the input layer,

$\mathbf{w}_{21}$ is the weight between the output layer and hidden layer,

4.3.2. BP Algorithm Equations

Given that the weights and biases have been initialised to small random values, the BP algorithm equations can be described as follows for a single input/output pair ($\mathbf{i}_0$, $\mathbf{t}$) where $\mathbf{t}$ is the target value:

Step 1 - Propagate the input signal $\mathbf{i}_0$ forwards through the network to obtain the output $\mathbf{v}_2$.

Step 2 - Compute the error (known as the energy function) for a given cost function the error is:

$$\mathbf{e} = \frac{\mathbf{1}}{\mathbf{2}(\mathbf{t} - \mathbf{v}_2)^2}$$

Step 3 - Compute the gradient of the error with respect to the net input to the output unit

$$\frac{\mathbf{de}}{\mathbf{dh}_2} = \mathbf{g}'(\mathbf{h}_2)(\mathbf{t} - \mathbf{v}_2)$$

for the output-to-hidden connection the gradient differentiation gives the following:

$$\Delta w_{21} = -\frac{de}{dw_{21}} = -\frac{de}{dh_2}\frac{dh_2}{dw_{21}}$$

$$= -\frac{de}{dh_2 v_1}$$

$$= \Delta b_2 = -\frac{de}{db_2} = -\frac{de}{dh_2}\frac{dh_2}{db_2}$$

$$= -\frac{de}{dh_2}$$

Step 4 - Compute the gradient of the error with respect to the net input to the hidden unit by propagating the error of the output unit backwards:

$$\frac{de}{dh_1} = -(t - v_2)\frac{dv_2}{dh_1}$$

$$= -(t - v_2)\frac{dv_2}{dh_2}\frac{dh_2}{dh_1}$$

$$= \frac{de}{dh_2}\frac{dh_2}{dh_1}$$

now

$$\frac{dh_2}{dh_1} = \frac{dh_2}{dv_1}\frac{dv_1}{dh_1}$$

$$= w_{21} g'(h_1)$$

hence

$$\frac{de}{dh_1} = g'(h_1) w_{21}\frac{de}{dh_2}$$

then

$$Dw_{10} = -\frac{de}{dw_{10}} = -\frac{de}{dh_1}\frac{dh_1}{dw_{10}}$$

$$= -\frac{de}{dh_1 i_0}$$

$$Db_1 = -\frac{de}{db_1} = -\frac{de}{dh_1}\frac{dh_1}{db_1}$$

$$= -\frac{de}{dh_1}$$

Step 5 - For each connection, w_{21} for example, update the connection (**Fig 4.4**).

$$w_{21} = -w_{21} + \Delta w_2$$

4.3.3. The Implementation

In the incremental model the above operations are performed for each input/output pair. This completes a sweep of all patterns. The next step is to go to (**step 1**) and repeat until the desired accuracy is attained (testing method). In the batch model, the weight and bias error derivatives are assimilated for all input/output pairs (**step 5**); the updating takes place after a complete sweep of all patterns (training method). For gradient descent apropartionlity factor, the so called learning-rate or "step-size", is usually incorporated to adjust the size of the increment which equals 0.5 because the output is less certain (i.e. close to 0.5) than those in which it is more certain (i.e. close to 0 or 1). In general, we could describe all the **BP** algorithm equations (rules) in terms of the following:

$$a_{pj} = \sum_{i=0}^{N-1} \left(w_{ji}\, o_{pi} + u_j \right)$$

where

a_{pj} is the activation of the j^{th} unit for the p^{th} training,

i is the input unit,

w is the weight,

u is the threshold or bias,

o is the output unit.

p is the activation of the i^{th} unit for the p^{th} training.

The error, measured by the value of a_{pj} is used to determine the weight changes, the object here is to find a way of attributing weights for neural layers that have an output function more broadly specified than the above:

$$o_{pj} = f_j (a_{pj})$$

Compute the changing of the weight:

$$\Delta_p w_{ji} = \beta \partial_{pj} o_{pi} \qquad (2.1)$$

where β is constant (learning-rate = 0.5),

∂_{pj} is the derivation of the error.

Compute the error (∂_{pj}) in fact the error is stated as:

$$\partial_{pj} = (t_{pj} - o_{pj}) f_j(a_{pj}) \qquad (2.2)$$

the term (t_{pj}-o_{pj}) clearly indicates that the error is proportional to the difference between the actual output o_{pj} and the target output t_{pj} the term $f_j(a_{pj})$ means the "rate of change of o_{pj} with respect to $f_j(a_{pj})$". This last point is illustrated for a particular form of $f_j(a_{pj})$ in **(Fig 4.5).**

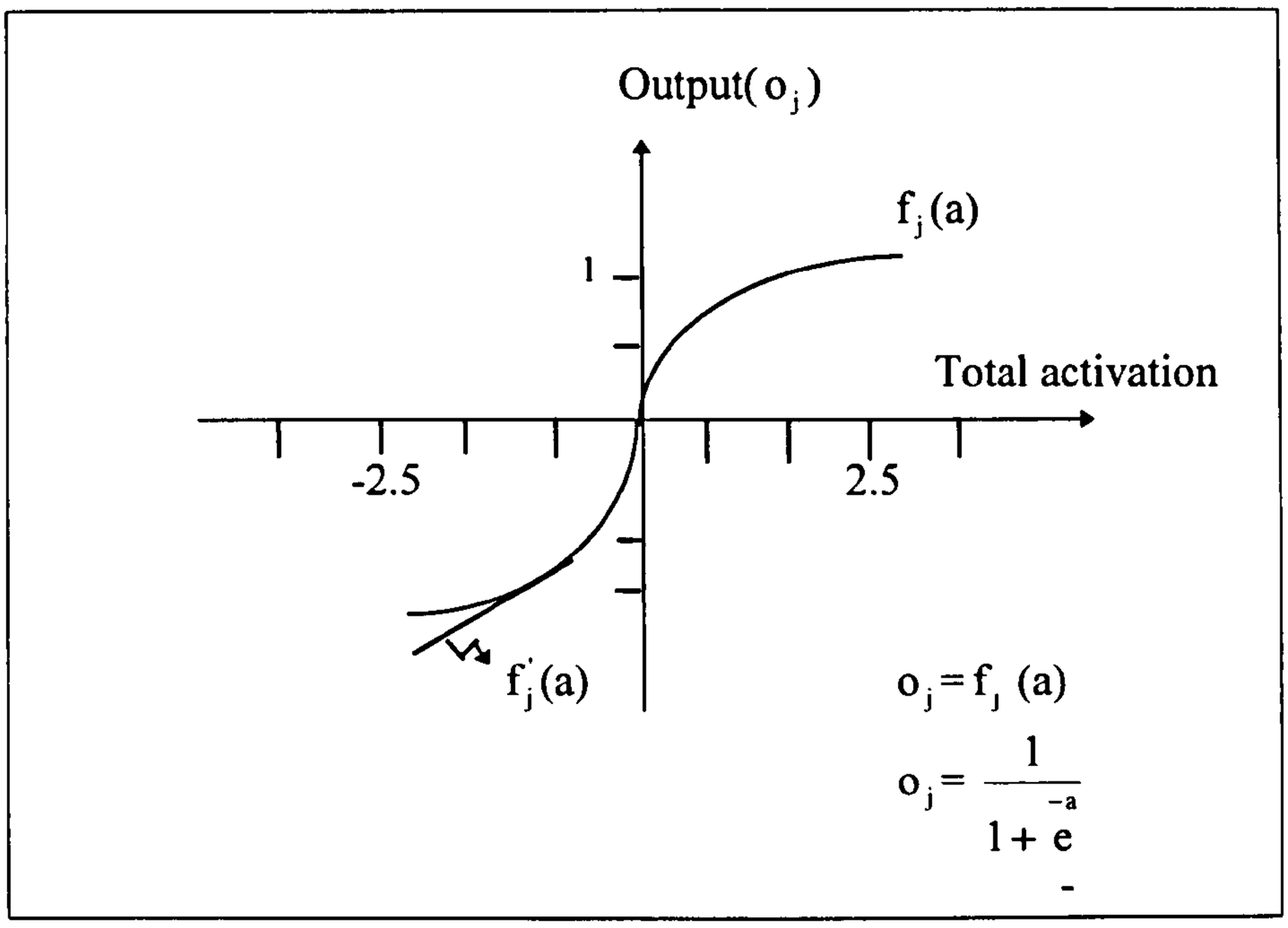


Fig 4.5 Sigmoid Function

Therefore $\partial_{pj} = \left(\sum_{k=0}^{N-1} \partial_{pk} w_{kj} \right) f_j'(a_{pj})$ (2.3)

where k is hidden unit connected to the output unit.

4.3.4. The Two Steps Methods for BP Algorithm

From the **BP** algorithm the training implies two steps (methods) [13], first is the **"forward step"** during which the input is applied and allowed to propagate to the output. The error values of the output units are calculated and compared to the targets for such units (which must be known). During the second step **"backward step"** phase these errors are propagated backwards and weight changes made to be precise, once the output error has been calculated, the weight changes in the output layer can then be made using (**equation 2.3**) and (**equation 2.1**). This fixes the error values for the previous layer using (**equation 2.3**) and the weight changes can be made using (**equation 2.1**) in the first inner layer. This procedure may be propagated backwards, the weights in the input unit are adjusted followed by another **"forward step"** and future **"backward step"** and so on.

The error is computed by the following equation:

$$E_p = 0.5\sum_{j=0}^{N-1}\left(t_{pj} - o_{pi}\right)^2 \qquad (2.4)$$

which is minimised by the two-steps error **BP** algorithm (Rumelhart, **1986**)[12].

4.3.5. Convenient Activation Functions for the BP Algorithm

The S-shaped function as shown in (**Fig 4.3**) is useful in this sort of work where

$$o_{pj} = \frac{1}{1+e^{-a_{pi}}} \qquad (2.5)$$

$$a_{pj} = \sum_{i=o}^{N-1} w_{ji} o_{pj} + u_j \qquad (2.6)$$

and

$$\mathbf{f}_j'(\mathbf{a}_{pj}) = \mathbf{o}_{pj}(\mathbf{1} - \mathbf{o}_{pj}) \qquad (2.7)$$

This simplifies the weight adjustment rules for output layers, (**equation 2.2**) becomes:

$$\partial_{pj} = (\mathbf{t}_{pj} - \mathbf{o}_{pj})\mathbf{o}_{pj}(\mathbf{1} - \mathbf{o}_{pj}) \qquad (2.8)$$

and for hidden layers (**equation 2.3**) becomes:

$$\partial_{pj} = \left(\sum_{k=0}^{N-1} \partial_{pk}\, \mathbf{w}_{kj}\right)\mathbf{o}_{pj}(\mathbf{1} - \mathbf{o}_{pj}) \qquad (2.9)$$

4.4. Operations for Image Processing

4.4.1. Introduction

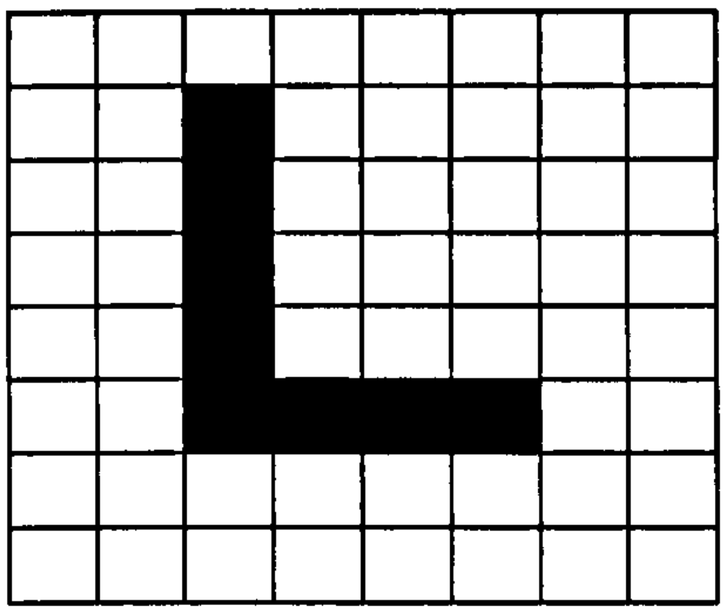
The interpretation of images requires a method that should serve two specific purposes. First it should encapsulate information which defines the important characteristics of the image. Second it should provide a basis for convenient and efficient processing by computer [8].

4.4.2. Image Representation

The simplest representation to generate, store and manipulate an image is a two-dimensional array. For our purpose we will use a Binarzed image which consists of just two possible level 1 or 0 in the resulting image. An array of points (8x8 for example) which is each either completely black (represented by 1) or completely white (represented by 0).

The compression of an array of two-valued pixels is known as a Binarzed image (**Fig 4.6(a)** and **Fig 4.6(b)**).

where $\begin{cases} 0 & \textit{if pixel i is white} \\ 1 & \textit{if pixel i is black} \end{cases}$



0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig 4.6(a) Grey image **Fig 4.6(b) Binarzed image**

4.4.3. Image Data Compression

Un-Preprocessing Image

The un-preprocessed image is shown in (**Fig 4.7**).

0	0	0	0	0	0	0	0	→ 0
0	0	1	0	0	0	0	0	→ 1
0	0	1	0	0	0	0	0	→ 1
0	0	1	0	0	0	0	0	→ 1
0	0	1	1	1	1	0	0	→ 4
0	0	0	0	0	0	0	0	→ 0
0	0	0	0	0	0	0	0	→ 0
↓	↓	↓	↓	↓	↓	↓	↓	
0	0	5	1	1	1	0	0	

Fig 4.7 Un-Preprocessed Image

Pre-processing Image

To make the binarzed image suitable for Neural Network recognition purposes, the binarzed image must be converted into real decimal values between 0 and 1 each value representing a column and row of the binarzed image, such that all columns and rows respectively will be an input to each node of the input layer of the Artificial Neural Network model [8], using the following equations:

$$\mathbf{I_j} = \sum_{i=0}^{N-1}\left(\sum_{j=0}^{N-1}\mathbf{image_{ij}}\right) \Big/ \mathbf{N^2}$$

The algorithm is

1-Repeat (for row processes)

i=0;

Add all the 1's of i^{th} row of the binarzed image, divided by N^2 and store in the i^{th} input node of the input layer.

i=i+1;

until (i > N);

The result of the above algorithm is shown below

column-sum: 0.00 0.00 0.10 0.11 0.12 0.13 0.00 0.00

$$\mathbf{I_i} = \sum_{i=0}^{N-1}\left(\sum_{j=0}^{N-1}\mathbf{image_{ji}}\right) \Big/ \mathbf{N^2}$$

Again, the algorithm is

2-Repeat (for column processes)

j=0; i=0;

Add all the 1's of the j^{th} column of the binarzed image, divided by N^2 and stored in the j^{th} input node of the input layer.

i=i+1; j=j+1;

until (j > max size);

The result of the above algorithm is shown below

row-sum: 0.00 0.00 0.10 0.11 0.13 0.12 0.10 0.00

where

column-sum is representing all 1's in the columns of the image (array) divided by $\mathbf{N^2}$,

row-sum is representing all 0's in the rows of the image (array) divided by $\mathbf{N^2}$,

$\mathbf{N^2}$ is the size of the image (array).

4.5. Data Transformations to Normalize the Image

4.5.1. Introduction

Automatic image interpretation procedures require the implementation of data transformations which operate not so much to change the inherent image characteristics, but rather to normalise certain properties which are significant in this particular work. It will be necessary to use some data transformation algorithms to normalise the image [8].

4.5.2. Position Normalisation Algorithm

The algorithm works well on a binarzed image and it is used to find the centre of the object within the image boundaries $\mathbf{N^2}$. This requires the determination of the X-axes and Y-axes of the image, which gives the co-ordinates of the centre of the object (X*,Y*). The algorithm consists of implementing the following equations :

$$\mathbf{X^*} = \frac{\sum_{x=0}^{x=N-1}\left(\sum_{y=0}^{y=N-1} \mathbf{x\ f(x,y)}\right)}{\sum_{y=0}^{y=N-1}\left(\sum_{x=0}^{x=N-1} \mathbf{f(x,y)}\right)} \qquad (1)$$

$$Y^{*} = \frac{\sum_{y=0}^{y=N-1}\left(\sum_{x=0}^{x=N-1} y\ f(x,y)\right)}{\sum_{x=0}^{x=N-1}\left(\sum_{y=0}^{y=N-1} f(x,y)\right)} \qquad (2)$$

4.5.3. Rotation Algorithm

To obtain a best fit of the object it may be required to rotate the object in the image $(x,y)\rightarrow(x',y')$; the rotation is defined by the following equations:

$$\mathbf{x'=x\cos(\theta)-y\sin(\theta)} \qquad (1)$$

$$\mathbf{y'=y\sin(\theta)+y\cos(\theta)} \qquad (2)$$

where

(x',y') is a transformed pixel in the image,

(x,y) is the original pixel in the image,

θ is the rotation angle.

The algorithm is implemented as follows:

1. Read a point (x,y) from the object;

2. Apply the equation (1) and equation (2) to obtain the rotating point (x′y′) of the object in the image;

3. Restore the new points in the image the nearest neighbour approximation.

The result of implementing the rotation algorithm gives poor quality when interpolation is applied.

References

[1] **A Roberts, M Yearworth**, *Comparison of Preprocessing Transforms for Neural Network Classification of Character Images*. Bristol Polytechnic, UK. PP 189, Image Processing and Its Applications.

[2] **P Flocchini, G Mauri, F Gardin, M P Pensini, P Stofella**, *Using Structured Input Patterns for Neural Based Image Recognition*, Universita di Milano, Italy, PP 213, Image Processing and Its Applications.

[3] **G D Kendall, T J Hall**, *Performing Fundamental Image Processing Operations Using Quantified Neural Networks*, King's College London, UK, PP226, Image Processing and Its Applications.

[4] **Tian-Jin Feug, Z Honkes, M J Korsten, L J Spreeuwers**, *Internal Measuring Models in Trained Neural Networks for Parameter Estimation from Images*, Ocean University, P R China, Twenty University, The Netherlands, PP 230, Image Processing and Its Applications.

[5] **Brian W Kerningham, Dennis M Ritchie**, *The C Programming Language*, Second Edition, Prentice Hall.

[6] **Al Kelley, Ira Pohl**, *A Book in C*, Second Edition.

[7] **Adam Blum**, *Neural Networks in C++*, 1992, Wiley & Sons, inc, bp 6 - 8

[8] **Michael C, Fairhurst**, *Computer Vision for Robortic Systems An Introduction*, Prentice Hall, 1988.

[9] **Stephen Grossbery**, *Neural Networks and Natural Intelligence*, bp 232 - 236, 1988.

[10] **A Philip, R Nachstheim**, *A Stable Second Order Method for Training Back-Propagation Neural Networks*, International Science Division (NASA ARC), Moffett Field, CA 94035, pp 452-455.

[11] **Igor Aleksander & Hellen Morton**, *An Introduction to Neural Computing*, Chapman and Hall, 1990.

[12] **Vallurn B Rao, Hayagriva V Rao**, *C++ Neural Networks and Fuzzy Logic*, BP 103, MIS, 1993.

[13] **Stephen Grossberg**, *Neural Networks and Neural Intelligence*, 1988, pp232-240.

Chapter 5

Data Image Compression Techniques

This chapter discusses the different data image compression techniques that have been researched .Each technique is supplemented with appropriate results.

In this chapter **Page No**

5.1. Introduction

The primary purpose of this chapter is to discuss various data compression techniques. Data compression seeks to reduce the size of the image data file. This chapter does not take a comprehensive look at every variety of data compression. It has been written to cover are the Image Data Compressions that have been implemented in the DIP software package, which works for greyscale images (not color images), and the decompression techniques. Each technique is used supplemented with results. The Data Compression techniques used here are lossy compressions, except the Differential Compression technique which is lossless.

5.2. Pixel Average Compression Algorithm

The Pixel Average Compression (PAC) is the simplest algorithm to reduce (compress) the image size. The PAC algorithm does not have complex mathematical equations and it is very fast. However, it is a lossy compression technique.

5.2.1. Mathematical Background

- The PAC uses is a simple approach to computing the mean (average) of a block of pixels. The formula is shown in (**Fig 5.1**).

$$\text{mean} = \frac{\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} \text{pixel}\,(i, j)}{NxN}$$

where N is the Block size

Fig 5.1 The mean equation

- Compute the size of the reduced image using the equation shown **(Fig 5.2).**

$$\text{reduced image size} = \frac{\text{Image size}}{\text{Block size}}$$

Fig 5.2 The reduced-image-size formula

5.2.2. Steps Analysis

The PAC algorithm works as following:

Step 1: Divide the image into blocks of size 4,8,16 or 32 (depending upon the user options) which represent the PAC Compression ratio.

Step 2: Compute the mean (average) of each block and round it to the nearest integer.

Step 3: Save (write) the mean of the block into a corresponding pixel of the reduced image.

Step 4: Take the next block of the image from left to right, top to bottom.

Step 5: Repeat Steps 2 and 3 until there are no more blocks in the image.

5.2.3. Examples and Results

Fig 5.3(a) Original Mona (128x128)

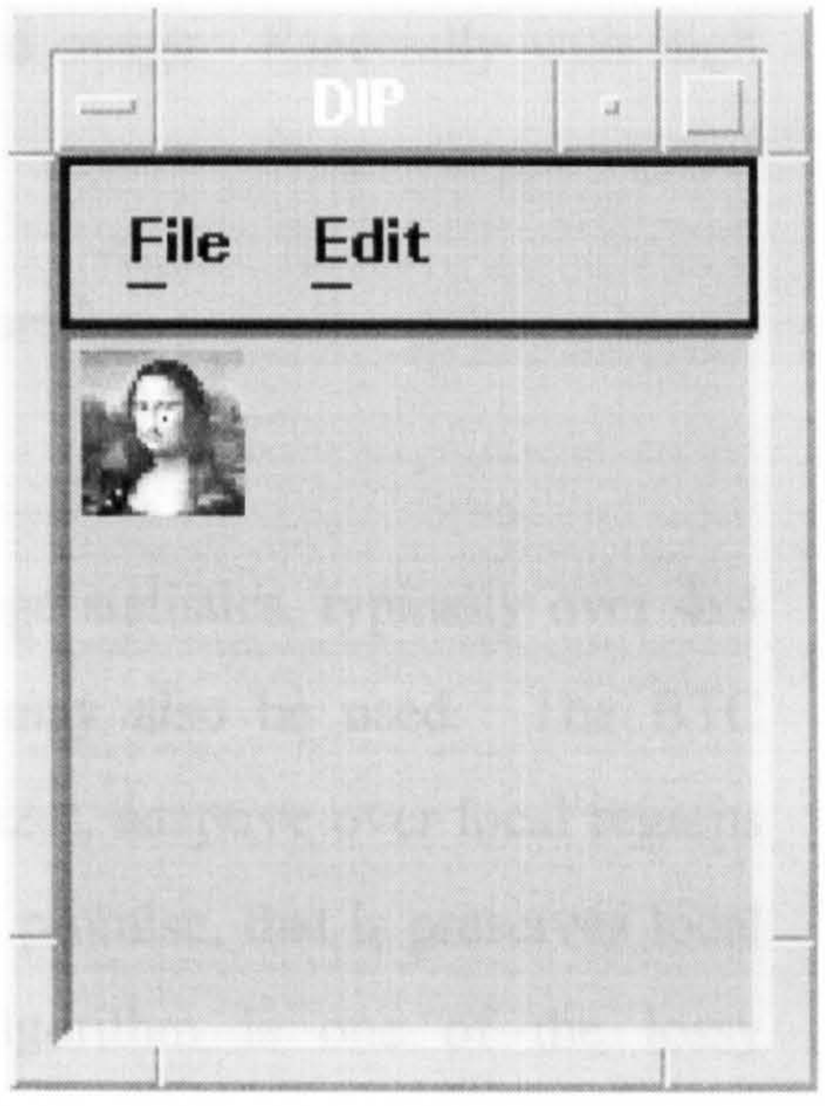


Fig 5.3(b) PAC Decompressed Image

Fig 5.4(a) Original Lena (128x128)

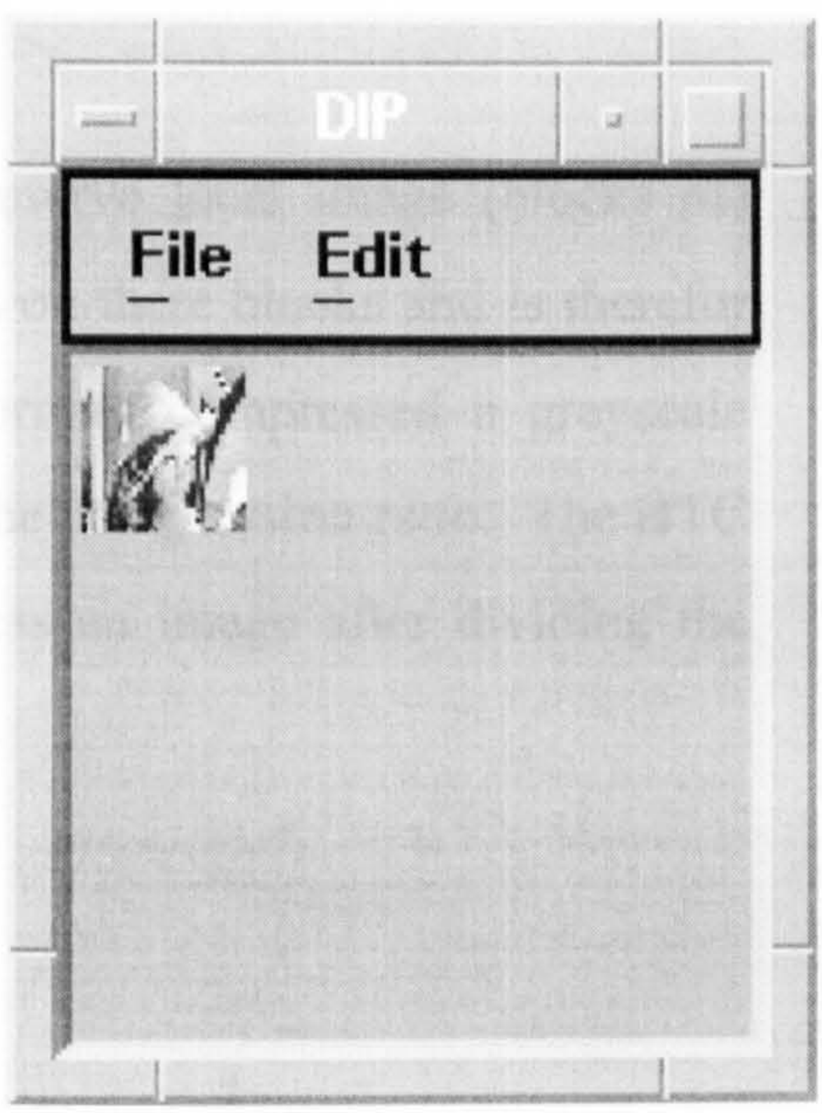


Fig 5.4(b) PAC Decompressed Image

5.2.4. Discussion

The PAC algorithm is very simple to implement and reduces the image, but does not give a very good quality compressed image. Especially with high compression ratio.

5.3. Block Trunction Compression Algorithm

5.3.1. Introduction

The idea behind BTC is to preseve local image statistics, typically over 4x4 blocks of the image, though larger blocks may also be used. The BTC technique has a one-bit non-parametric quantizer, adaptive over local regions of the image. The quantizer that shows great promise, that is preserves local statistics of the image [1]. The BTC algorithm is one of the lossy compressions group, examples of this group are the Fourier and Cosine Transform Compression algorithms.

5.3.2. Mathematical Background

The idea behind the BTC algorithm is to preserve local image (blocks of) typical size 4x4, so that the algorithm operates on there blocks and is therefor fairly simple and quite fast. The BTC algorithm compressed a greyscale image to one quarter of original size which is the compression ratio. The BTC algorithm uses the following steps to compress an image after dividing the image into NxN blocks, each block of size 4x4.

Step 1: Compute the mean

$$\mathbf{m_1 = \frac{1}{n} \sum_{i=0}^{n-1} P_i}$$

where P = pixel, n = 4

Step 2: Compute the mean of the squares of the pixel values

$$m_2 = \frac{1}{n}\sum_{i=0}^{n-1} P_i^2$$

Step 3: Let q be the number of pixels greater than or equal to the mean, then compute a and b.

for all q, q = 0, 1, ..., 15
if q ≥ m_1 output = b
if q < m_1 output = a

The mean becomes

$$m_1 = \frac{1}{n}\left[(n-q) * a + q * b\right]$$

Step 4: The second mean becomes

$$m_2 = \frac{1}{n}\left[(n-q) * a^2 + q * b^2\right]$$

Because we want to preserve the mean and the second moment, we require these values to be the same as the original values. Thus, the equation in (step 1) is equal to that in (step 3) and the equation in (step 2) is equated to that in (step 4).

Step 5: Compute the outputs a and b

$$a = m_1 - s * d$$
$$b = m_1 + s / d$$

Step 6: Compute the variable **d** that is needed in (step 5)

$$d = \sqrt{\frac{q}{(n-q)}}$$

Step 7: Compute the variable σ that is needed in (step 5)

$$s = \sqrt{m_2 - m_1^2}$$

5.3.3. Step Analysis

The BTC algorithm operates according to the following steps:

Step 1: Divide the image into blocks of 4x4 pixels. For each block, compute the mean (m_1) and the second moment (m_2).

Step 2: Construct a bit plane by replacing each pixel that has a value less than the mean with a 0, and pixels with values greater or equal to the mean with a 1.

Step 3: Pack the resulting bit plane into 2 bytes which becomes the block trunction code.

Step 4: Write a, b and the 2-byte code to the output file.

The ***decompression*** stage is straightforward.

Step 1: Read a, b and a 2-byte block trunction code from compressed file.

Step 2: Examine each bit in the 2-byte code. If a bit is clear, then display (or write) the value a. If a bit is set, display the value b. Repeat.

Step 3: Repeat Step 2 until no more codes are in the file.

5.3.4. About the BTC Algorithm

The BTC decompressed file is different from the original, but generally quite good for 4x4 block sizes. The 4x4 BTC is not an error-free compression algorithm and any image requires only four times less disk space than the original. Finally, what happens when the BTC algorithm is applied to an image that has gone through the compress/decompress cycle once before - is there further reduction in image quality? The answer is simply No.

5.3.5. Examples and Results

Fig 5.5(a) Original Mona (128x128x8)

Fig 5.5(b) BTC Decompressed Image

Fig 5.6(a) Original Lena (128x128)

Fig 5.6(b) BTC Decompressed Image

5.4. JPEG Image Compression Technique

5.4.1. Introduction

The new international standard for image compression is called JPEG. JPEG is an acronym for "Joint Photographic Expert Group". JPEG now stands at the threshold of widespread use in diverse applications. Many new technologies are converging to help make this happen. High-quality continuous-tone greyscale or color display are now a part of most personal computing systems. However, until JPEG compression came upon the scene, the massive storage requirement for large number of high-quality images was a technical impediment to widespread use of images [2]. JPEG has provided a high-quality yet very practical and simple solution to image compression and its lossy technique.

5.4.2. Mathematical Background

5.4.2.1. The Discrete Cosine Transform (DCT)

The DCT is one of the basic building blocks for JPEG and it is the key to the compression process. The DCT is in a class of mathematical operations that includes the well-known Fast Fourier Transform (FFT), as well as many others. This transformation is done frequently when analysing digital audio samples using the FFT. The FFT transforms the set of samples points into a set of frequency values that describe exactly the same signal.

$$DCT(i,j) = \frac{1}{\sqrt{2N}} C(i) C(j) * \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} pixel(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

Fig 5.7 The Discrete Cosine Transform (DCT)

$$\text{pixel}(x,y) = \frac{1}{\sqrt{2N}} * \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} C(i)\,C(j)\,DCT(i,j)\cos\left[\frac{(2x+1)i\pi}{2N}\right]$$

$$\cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

where

$$C(i), C(j) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } i,j = 0 \\ 1, & \text{if } i,j > 0 \end{cases}$$

Fig 5.8 The Inverse DCT (IDCT)

The actual formula for the 2-D DCT is shown in (**Fig 5.7**) with its partner, the IDCT (used for decoding), shown in (**Fig 5.8**). The two functions shown above can be computed with a relatively straightforward piece of code.

A very important point to make about this type of transformation function (DCT) is that, the function is *reversible* (IDCT).

The time required for each element in the DCT is heavily dependent on the size of the matrix NxN. To get around this problem DCT implementations typically divided the image down into smaller blocks. The JPEG group selected an 8x8 block for the size of their DCT quantization, which become standard for the image processing., and to be compatible for practical implementations using today's technology. This type of compression is called "block coding",

5.4.3. The Algorithm Stages

The JPEG Compression/Decompression for greyscale images algorithm operates in four successive stages, shown in (**Fig 5.9**).

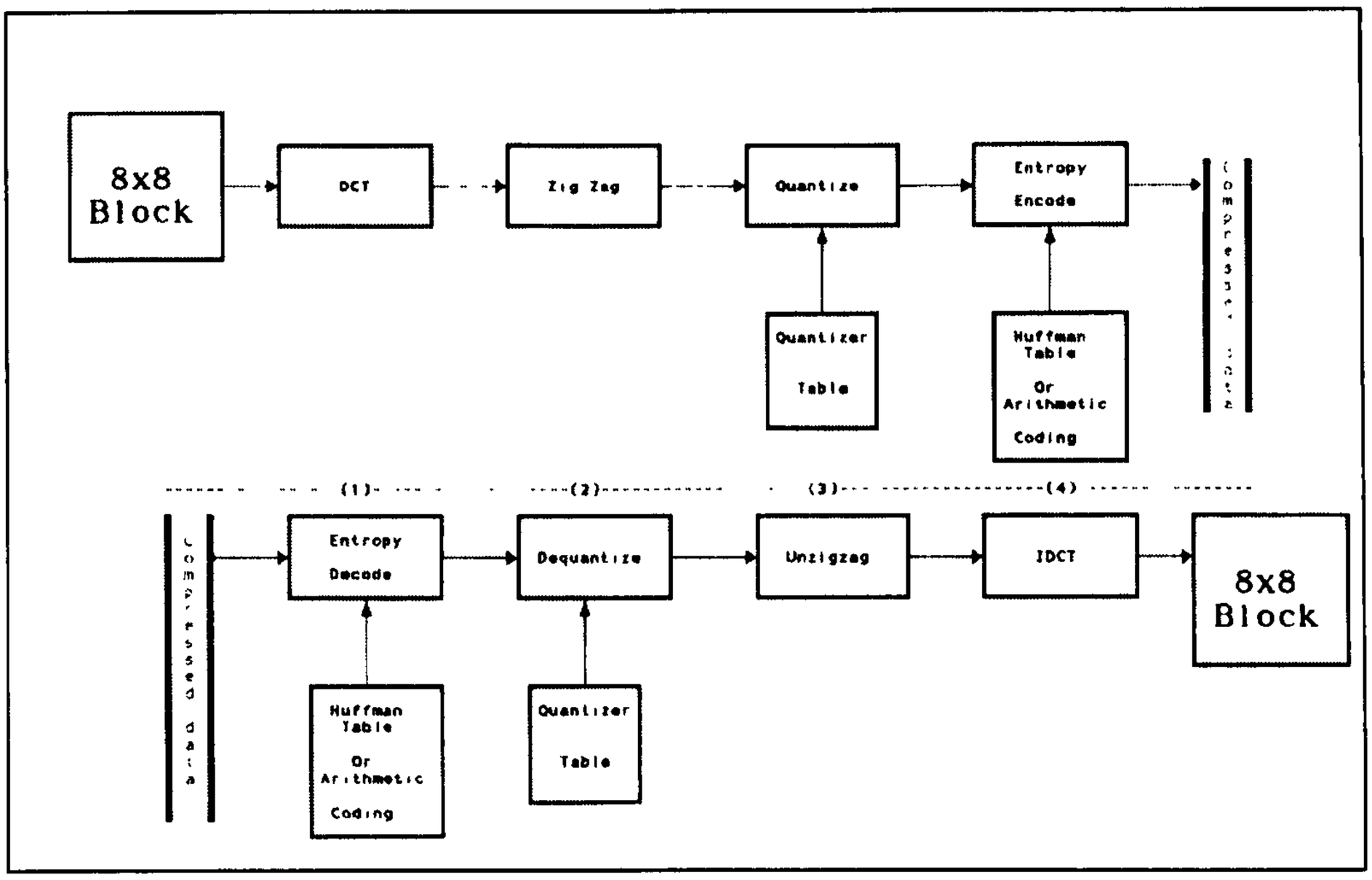


Fig 5.9 The JPEG Compression / Decompression Stages

During encoding (compressing), an image is broken down into 8x8 pixels blocks, that are processed individually, from left-to-right and top-to-bottom. Each block of the image data is subjected to a forward discrete cosine transform (DCT), which converts pixel values into their corresponding frequency. The frequency component values, or coefficients are scanned "zigzagged" in order of increasing frequency. The resultant frequency coefficients are then quantized, which causes components with non-zero amplitudes to become zero. The level of frequency-component quantization depends upon a user-specified image-quality factor.

The entropy encoding process performs two types of image - compression on the block of frequency coefficients.

First, it run-length encodes the number of zero-coefficient values preceding a non-zero coefficient in a block. Secondly, it bit encodes the coefficient values

using statistically generated tables. The encoded bit stream is written to the output stream or file.

During decoding (decompression), the encoded bit stream is read from the input stream or file, and the quantized frequency coefficients for a block of pixel data are decided. These frequency components are then dequantized by multiplying them by the quantization table with which they were produced. The frequency coefficients are scanned from frequency order back into pixel order. The block of frequency components is then subjected to an Inverse Discrete Cosine Transform (IDCT), which converts the frequency component values back into pixel values. The reconstructed pixel values are stored in the decoded image [3].

A more detailed discussion of this approach is given below:

- *The first stage*. The DCT transformation produce frequency coefficients which identifies pieces of information in the image that can effectively "thrown away" without seriously compromising the quality of the image.

- *The second stage*. Quantization simply means the process of reducing the number of bits needed to store an integer value (pixel) by reducing the precision of the integer. Once the image has been compressed, we can generally reduce the precision of the coefficients more and more as we move away from the DC coefficient at the original.

- *The third stage*. The JPEG algorithm implements the quantization using a quantization matrix. The function used for quantization is shown in (**Fig 5.10**) that rounds the output to the nearest integer.

$$\text{Quantized value } (i,j) = \frac{\text{DCT}(i,j)}{\text{Quantum}(i,j)}$$

Fig 5.10 Quantization function

The inverse or dequantization of the above function is shown in (**Fig 5.11**) and used for the decoding process.

$$\text{DCT } (i.j) = \text{Quantized value } (i,j) * \text{Quantum } (i,j)$$

Fig 5.11 The Dequantization function

- *The final stage*. In the JPEG process is coding or compression. The JPEG coding combines three different steps to compress the image.

Step 1: Changes, the DC coefficient at (0,0) from an absolute value to a relative value, i.e. coding the DC element as the difference from the previous DC element which typically produces a very small number.

Step 2: The Zig-Zag Sequence. The zig-zag sequence is the method to arrange the coefficients of the image (**Fig 5.12**). The reason the JPEG algorithm is so effective is that a large number of coefficients in the DCT image are truncated to zero values during this problem.

Step 3: Encoding using two different encoding mechanisms. The first is the run-length encoding of zero-values. The second is the entropy encoding which involves either the Huffman codes or arithmetic coding. Further details about these two methods are given below.

Quantization stage

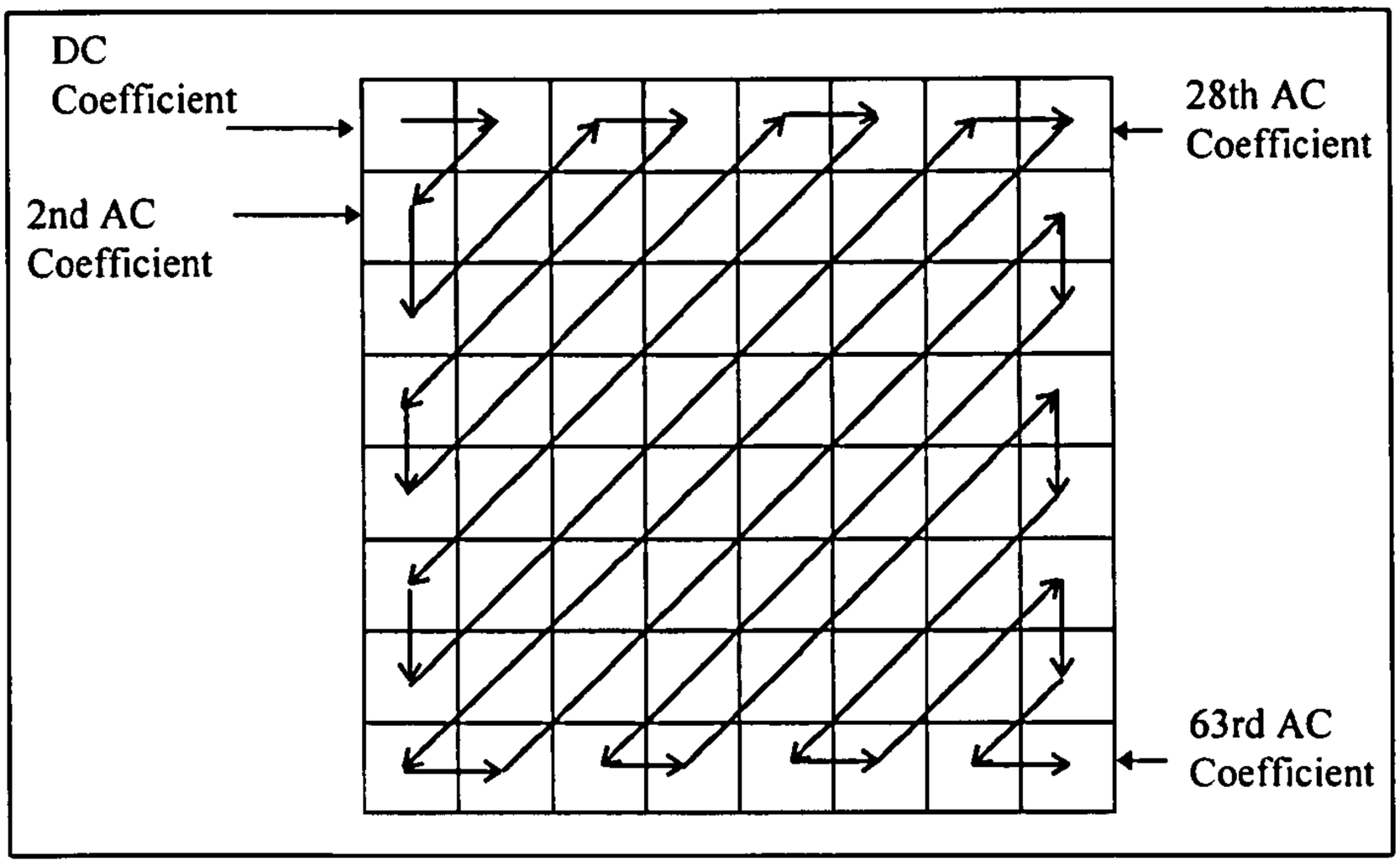


Fig 5.12 The Path of the Zig-Zag Sequence in 8x8 Block

- **The DCT Coefficient Arithmetic Coding**

The arithmetic coding model deals with the zero coefficients explicitly, rather than coding runs of zero coefficients. When a non-zero coefficient is encountered, the sign is coded, followed by the logarithm of the magnitude and then the low order bits of the magnitude. Following each non-zero coefficient an end-of-block decision is coded indicating whether the end of the block has been reached. The statistical model used for the binary decision sequence conditions are the most important decisions on the position in the zig-zag matrix. Conditioning on the sign and on the coefficient being low or high "frequency" is also used [4].

- **The Huffman Encode**

Using Huffman encoding provides additional lossless compression for already highly processed image data. It is based upon the statistical characteristics of the data to be compressed, symbols (bits) that occur frequently in the data are assigned shorter Huffman codes and those that occur infrequently and are assigned longer codes. The Huffman codes assigned to the various symbols are bit packed together into the tightest possible configuration of bytes.

5.4.4. Discussion

The JPEG Committee proposed the use of adaptive binary arithmetic or Huffman encoding for eutropy coding to the JPEG algorithm. It consistently achieves higher data compression and has the advantage of being a one-pass adaptive encode procedure. It is generally acknowledged that arithmetic encoding performs marginally better for some images, but is much more complex to implement; on the other hand, Huffman encoding is in the public domain and can be used without worry of patent infringement. Consequently, Huffman encoding is utilised in most JPEG implementations. Huffman encoding / decoding is difficult to write and debug because the stream data has to be examined at the bit level. JPEG compression is adaptable to both hardware and software implementation and to application needs (flexibility).

5.4.5. Some Implementations and Results

Fig 5.13(a) Original Mona (128x128)

Fig 5.13(b) JPEG Decompressed Image

Fig 5.14(a) Original Lena (128x128)

Fig 5.14(b) JPEG Decompressed Image

5.5. Differential Image Compression Algorithm

5.5.1. Introduction

Differential Image Compression suggests some degree of graphics dependence, after all, the word "image" in the title implies that graphics must be used. Differential image compression keeps track of only the information necessary to change the currently active block buffer to the next block buffer. The differential image compression uses the active fixed-size buffer for data and the compression/decompression algorithms simply modify the data in the buffer. There is nothing inherent in this approach to prevent or inhibit the buffer from being any portion of random-access memory containing any type of data.

5.5.2. Algorithm Steps Analysis

The implementation of differential image compression uses three steps as follows:

Step 1: The code is simple, which means the code structure is so easy that the entire data buffer can be kept in memory. The buffer contains exactly 256 blocks, each block having 8 bytes.

Step 2: The source data (image) is encoded by comparing successive 8 byte blocks with all 256 blocks in the data buffer. The index of the best matching block is then output as one byte and differential information is output to update the best matching block with the new data. Overall compression is optimised when the source data (image) contains similar, ideally repetitive, blocks of data.

Step 3: To guarantee at least one matching byte for the first block of source data, block #0 is filled with 0 bytes, block #1 is filled with 1 bytes, block #2 is filled with 2 bytes and so on to block #255, which is filled with all 255 bytes.

5.5.3. Coding Details

Here in brief are the instructions for the Differential Image Compression Algorithm.

a) The following instructions present the encoding (compression) algorithm:

1. *Initialise all blocks in data buffer.*
2. *Load one block of source data from the file.*
3. *Find best matching block in data buffer.*
4. *Output index of best matching block.*
5. *Output a bitmap for the ordered bytes in the data buffer which need to be changed.*
6. *Output all new bytes in loaded block of source data.*
7. *Update best matching block with loaded block of data.*
8. *Repeat from instruction 2 until there is no more data in the file.*

b) The next instructions present the decoding (decompression) algorithm.

1. *Initialise all blocks in data buffer.*
2. *Load best matching block index.*
3. *Load bitmap for the ordered bytes in the data buffer which need to be changed.*
4. *Directly load all new (i.e. changed) bytes into the best matching block of data.*

5. *Output the best matching block of data which now contains an exact copy of the originally loaded block.*

6. *Repeat from instruction 2 until there is no more data in the file.*

5.5.4. Some Examples and Results

Fig 5.15(a)

Original Lena

Fig 5.15(b)

Decompressed Lena

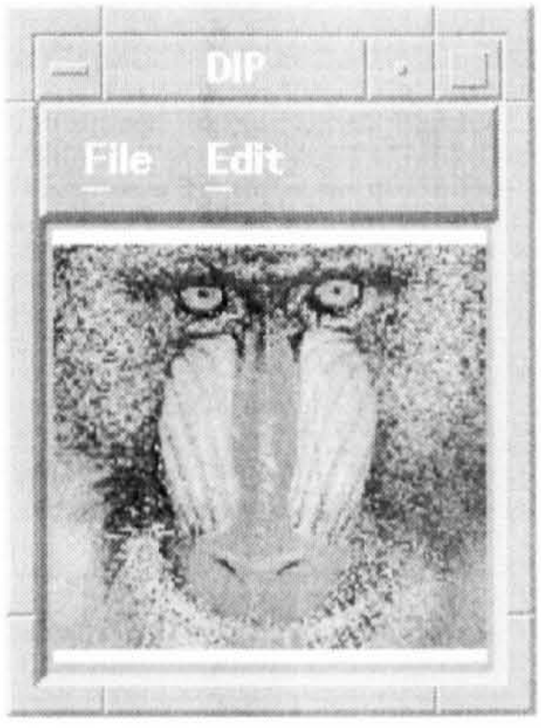


Fig 5.16(a)

Original Mand

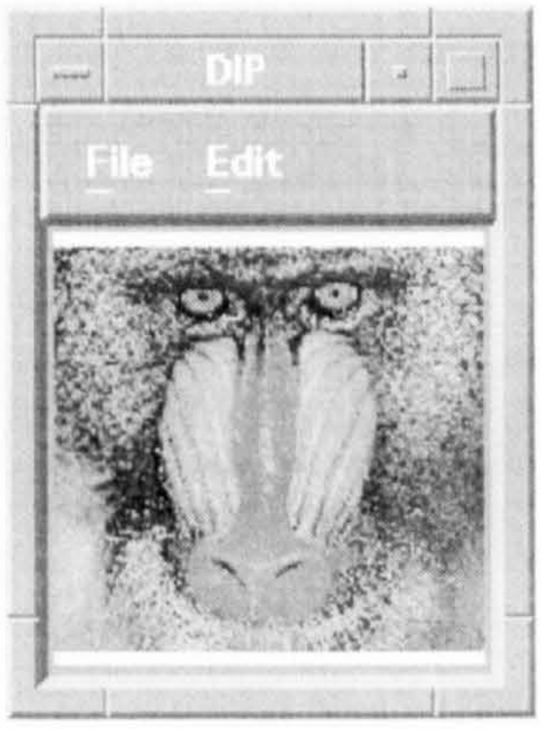


Fig 5.16(b)

Decompressed Mand

5.5.5. Discussion

The Differential Image Compression algorithm can work with fixed or variant blocks as (one improvement to the algorithm). The algorithm is lossless i.e. it gives an identical copy of the original image from the decompressed file.

References

[1] Anton Kruger, *"Block Trunction Compression"*, Dr Dobb's Journal, April, 1992.

[2] Pen baker, W.B. & Mitchell, J.L, *"JPEG Still Image Data Compression Standard"*, December 1992

[3] Craig A. Lindley, *"JPEG-Like Image Compression Part 1 "*, Dr Dobb's Journal , July, 1995

[4] Mark Nelson, *"The Data Compression Book"*, 1991

[5] James H. Sylvester, *"Differential Compression Algorithms"*, Dr Dobb's Journal, April, 1993

Chapter 6

Fractal Image Compression Technique

This chapter discusses Fractal Techniques. In the first section we discuss the Fractal Image Compression Technique which is one of the key research points in this thesis and has been used as a method of pattern recognition (as explained in the following chapter). This chapter also considers Fractal Dimension Segmentation Techniques which describes all the methods that have been implemented in the DIP software package.

Page No

6.1. Fractal Image Compression Technique

6.1.1. Introduction

This chapter presents another advantageous method for compressing an image known as ***Fractal Transform Compression***. Most images contain some amount of redundancy that can sometimes be removed when the image is stored and replaced when it is reconstructed, but this redundancy does not lead to high compression unless the redundancy can be detected, in which can the data can be greatly compressed [1]. Fractal objects are redundant objects, in the sense that they are made up of transformed copies of either themselves or parts of themselves. A fractal scheme has been developed by M Barnsley, who founded a company based on fractal image compression technology but who has released only some details of his scheme [2]. A Jacquin, a former student of Barnsley's, was the first to publish a fractal image compression scheme [3].

The goal of this introduction is to explain Fractal Image Compression in very simple terms, with as little mathematics as possible and show some implementation results from images as fractals and images as non fractals.

6.1.1.1. Fractals

Imagine a special type of machine that reduces the image by half and reproduces it four times on the output, a machine (**Fig 6.1**) which can take the output back as input. After several iterations of this process, we observe that all the images seem to be converging to the final image; we also see that this final image is not changed by the process since it is formed from four reduced copies of itself.

It must have detail at every scale - it is a fractal. This is done by an attractor which reduces the input image, the copies of any initial image will be reduced to a point as we repeatedly feed the output back as input. There will be more copies and it gets smaller every time.

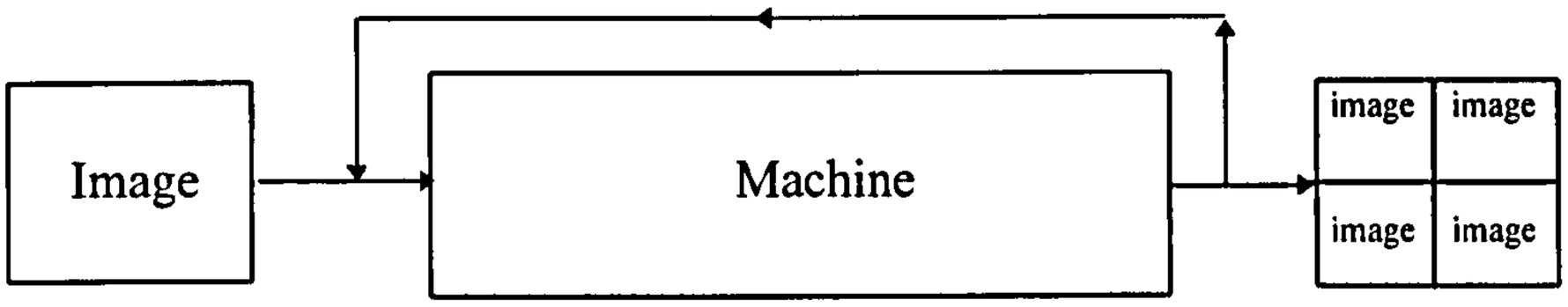


Fig 6.1 The Fractal Machine

Some examples of real world fractal images are clouds, trees, grass and the sea surface.

6.1.1.2. Fractal Image Compression

From the above, the final result of using this machine in a feedback loop is determined by the way the input image is transformed. Different transformations lead to different attractors. A given transformation applied to any two points in the input image must bring them closer together in the output. The result is a reduced image "compressed" **(see section 6.1.2.3.)**. The transformation can have any form and can have any attractor **(see section 6.1.2.1.)**.

6.1.1.3. Iterated Function System (IFS)

An IFS is just a function. In the context of images, the input to the function is an image and the result is another image. An IFS encodes an image by arranging for the image to be the unique fixed point of the IFS.

That is, the IFS is constructed so that the image that we wish to compress is invariant under the IFS mapping, the image is mapped to itself. It is here that the self referential or fractal nature of the compressing process arises. To construct or decode the image, all that is necessary is to iterate the IFS from any starting image and the compressed image will gradually appear as the iteration proceeds. Thus the decoding process is very simple, but what about the encoding process? How do we construct an IFS so that it has the image as its fixed point in the first place? In practice, we do not, but we do construct an IFS that has as its fixed point an image which is a close approximation to the original image.

6.1.2. Mathematical Background

6.1.2.1. Affine Transformations

The transformations that are used by the IFS are called Affine Transformations. In practice, choosing transformations of the form

$$W_i(\underline{x}) = W_i\begin{bmatrix}x\\y\end{bmatrix} = \begin{bmatrix}a_i & b_i\\ c_i & d_i\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}e_i\\f_i\end{bmatrix} = A_i\underline{x} + t_i$$

is introduced and it is sufficient to yield a rich and interesting set of attractors. Each affine transformation can skew, stretch, rotate and translate an input image. Each affine transformation is defined by six numbers a_i, b_i, c_i, d_i, e_i and f_i. An example of an affine transformations (maps) is shown below

$$W_i\begin{pmatrix}x\\y\end{pmatrix} = \begin{pmatrix} r\cos\theta & -s\sin\phi\\ r\sin\theta & s\cos\phi\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix} + \begin{pmatrix}h\\k\end{pmatrix}, \quad (i = 1,2,3,4)$$

where x,y are pixel co-ordinates
h,k are translations
ϕ,θ are rotation degrees
and r,s are the scalings

This is the IFS that generates the *Black Spleenwort fern* [2].

6.1.2.2. Iterated Function Systems (IFS)

In this section we will formally define iterated function systems. IFS's served as the motivating concept behind the development of fractal image compression, and most of the work on fractal image compression is based on IFS's and their generalisation. First, we will explain what the collage theorem is because it is central to the design of IFS's whose attractors are close to a given set. ***The Collage theorem*** states that if we find a contraction that approximately maps a digital image into itself then the fixed point of the contraction is an approximation to the original digital image. The result however holds more generally than this, although we have not yet said what we mean by a digital image or by contraction mapping on a set of digital images. With this in mind, we want to find a contractive transformation of an image that approximately maps it into itself. One way of doing this is to consider taking the original image and by a process of shrinking, skewing, grey level scaling and rotating, arrange that the transformed image visually matches a part of the original image. If this can be done for all parts of the image, that is, if we can cover the entire image with transformed parts of the original, then we have found a suitable transformation of the image to itself. The process of covering the image with transformed copies of itself is akin to producing a *collage* of the image [4]. More details are given in [2] and [4]. This theorem is central to the design of a mathematical model called an Iterated Function System (IFS). The collage theorem tells us how to find an IFS whose attractor is *close to* or looks *like* a given set. An IFS consists of a collection of contractive transformations which map the plane R^2 to itself (equation 1).

$$\{W_i : R^2 \to R^2 \mid i = 1, \ldots, n \qquad (1)$$

such that the union or collage of the images of the given set under the transformation is close to or looks like given set (equation 2).

$$\mathbf{W(x) = \bigcup_{i=1}^{N} W_i(x)} \qquad (2)$$

The map W is not applied to the plane, it is applied to sets, collection of points in the plane. Given an input set x, we can compute $W_i(x)$ for each i, take the union of these sets (this corresponds to assembling the reduced copies) and get a new set W(s) "image". The affine transformation used with IFS for grey scale fractal images is of the form

$$\mathbf{W_i = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & P \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} R_x \\ R_y \\ Q_i \end{bmatrix},}$$

where a_i,, b_i, c_i and d_i are the transformations act which in the xy plane.

P is the coefficient with a fixed positive number

such that $\phi < P < s$ and

$$V_i(z) = P_z + Q_i$$

6.1.2.3. The Hansdorff Metric

The Hansdorff Metric is used to compute the "distance" between any two points in the plane (image). We define the metric as a space. A metric space is a set of points along with a function *H* that takes two elements of the set and gives the distance between them. To find the Hansdorff distance h(A,B) between two subsets of the plane, A and B, we carry out the following procedure:

1. For each point x of A, find the closest point y in B.
2. Measure these minimal distances (both starting from A and starting from B).

3. Choose the largest one.

This is the Hansdorff distance. The following are some examples of metrics.

(a)

$$d((x_1,y_1),(x_2,y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

(b)

$$\max \{|x_1 - x_2|, |y_1 - y_2|$$

(c) the L^p metrics are

$$\|f - g\|_p = \left(\int_{R^2} |f - g|^p \, dxdy\right)^{1/p}$$

and there are more metrics [1].

6.1.3. Fractal Transform Algorithm Analysis

This section addresses the implementation of systematic realisation of the theoretical background described in (**section 6.1.2.**). Here we analyse the main stages of the algorithm in order to compress an image. All the work that has been done in this section is based on [5].

6.1.3.1. Image Partitions

Extract Domain Blocks and Range Blocks Stage

The original image is partitioned into non-overlapping squares of two different sizes. The larger squares called "Domain block" of size 8x8, are such that the domain blocks taken together must cover the original image. The domain blocks are numbered from 0 - 15 in order from left to right and from bottom to top. (**Fig 6.2**)

Domain 12	Domain 13	Domain 14	Domain 15
Domain 8	Domain 9	Domain 10	Domain 11
Domain 4	Domain 5	Domain 6	Domain 7
Domain 0	Domain 1	Domain 2	Domain 3

Fig 6.2 Domain Blocks over an image

The squares which are the smaller ones of size 4x4 are called "Range Blocks". Thus a partition constructed this way is image independent, it allows the coder

(i) to use large blocks to take advantage of smoothly varying image areas,

(ii) to use small blocks to capture details in complex areas (rugged boundaries, fine textures).

For the range block of image size 4x4 there are the following advantages:

a) they are easy to analyse and to classify geometrically,

b) they allow a fast evaluation of interblock distances,

c) they are easy to encode accurately and

d) they lead to a robust encoding system - one whose performance is steady even when source images are diverse.

6.1.3.2. The Multi-operation Stage

1. The shrinking operation

The method for producing shrunken range blocks will now be described in greater detail. This method is known as a ***direct shrinking*** method. A shrunken range block is the output of the method and range block is the input,

using shrinking by averaging method, in which a shrunken range block is obtained. The shrinking by averaging method may be expressed mathematically as:

$$shrunken\_range\_block(i,j) = (range\_block(2i,2j) + range\_block(2i+1,2j) + range\_block(2i,2j+1) + range\_block(2i+1,2j+1))/4$$

Unique identifiers are then assigned to the mapped range blocks. Each of the identifiers specify both the address of the subset of image data from which the corresponding range block was created and the procedure which was applied to mapped range block. In the preferred embodiment, the shrunken range blocks are assigned an identifier. The identifier includes an address and indication of affine transformation. Affine transformation has been applied to the range block from which the shrunken range block was derived. Each pixel of the domain blocks and the shrunken range blocks has a number associated with it which represents its greyscale value (level).

2. The Rotation Operation

The identifier, in the preferred embodiment also includes a first code indicating 0°, 90°, 180°, or 270° rotation, and a second code indicating whether inversion has been performed to produce a range block prior to shrinking.

6.1.3.3. Discrete Image Transformation

In this section, we describe the class of discrete contractive affine image transformations that the coder uses. Although addresses of range blocks in the preferred embodiment are x,y pixel co-ordinates, the encoding of the range

blocks consists in finding the **"closest"** (which means most similar to the domain block) such that the distortion is minimum. This alternative statement implies a method, which is useful in understanding the concept of comparisons between blocks of image data. That concept is referred to as **"distance".**

The Fractal Transform algorithm uses one of the Hansdorff distance [1],[6], which is known as the Euclidean Distance.

$$L^1 = \sum_{i=0}^{N-1} \left(A(i) - B(i)\right)^2$$

where A and B are image blocks.

The L^1 distance computes the squares of the differences between 2 blocks before they are summed in which case the "averaging" is achieved by taking the square root of the sum. To assist in finding a shrunken range block which closely corresponds to a domain block, vertical scaling parameters P and Q **(see section 6.1.2.2.)** may be applied to domain block pixel greyscale values to obtain a new pixel greyscale value.

Step 1:

Once the L^1 has been computed between the domain block and each of the shrunken range blocks, the shrunken range block that yields the lowest value of L^1 is selected as the "closest" shrunken range block to the specified domain block. If the L^1 just computed is less than the currently stored distance value, then the currently stored distance value is replaced by the distance value most recently computed. Therefore, the current shrunken range block values are stored and the most recently distance value computed is discarded.

The range block pointer (value) is incremented and a determination is made until all of the shrunken range block have been checked; if not, **step 1** is repeated until all shrinking range blocks have been checked.

Step 2:

The range block pointer (value) and the shrunken range block pointer (value) are then incremented. A determination is made to see whether all the range blocks have been shrunk, if not, **Step 1** is repeated until all range blocks have been shrunk.

Step 3:

The currently stored shrunken range block pointers (values), consisting of the x, y co-ordinates, and the associated procedure information (reflection and inversion) is then appended to the list of identifiers representing the domain block.

Step 4:

The domain block pointer (value) is incremented and **step 1** repeated until all domain blocks have been processed.

Step 5:

Finally, each domain blocks of the original image data is then output (written in a file), as a fractal transform of the image data representing the greyscale image.

6.1.3.4. The Decode Stage

In this section we will explain how a greyscale image is recovered from a fractal transform (which includes scaling parameters). An arbitrary initial image and a fractal transform are input to a decompression system. The initial image is loaded into a buffer A. A buffer B is divided into domain blocks and

into each domain block is read a shrunken range block which is specified by the corresponding address of fractal transform, scaled as specified by the vertical scaling parameters. The shrunken range block is constructed from initial image in buffer A. The image in buffer A is treated, addressed and processed into shrunken range blocks in exactly the same manner as the original image. Each of the identifiers forming the greyscale transform corresponds to one of the domain blocks in buffer B, which used to select a shrunken range block from buffer A. This is used to read into the corresponding domain block, thus forming an interim image in buffer B. The interim image in buffer B is now read back into buffer A and buffer B is cleared.

The steps of reading into each domain block of buffer B, the shrunken range block of buffer A specified by a corresponding address of the fractal transform, is used to form an interim image, and reading the interim image of buffer B into buffer A. These steps are repeated by a **prescribed** number of times.

The main steps for decompressing a greyscale fractal transform are as follows:

Step 1:

An initial image is supplied to buffer B.

Step 2:

A main loop counter, a domain block pointer for buffer B, and fractal transform pointer for buffer B are initialized.

Step 3:

Get the next entry of the fractal transform as specified by the fractal transform pointer.

Step 4:

A corresponding shrunken range block is created from buffer A, then scale the range block with vertical scaling parameters P and Q and read into a domain block of a corresponding position in buffer B.

Step 5:

Increment each of domain block pointer for buffer B and the fractal transform pointer. A determination is made whether the last domain block and fractal transform have been processed. If not, repeat back step 4 and step 5 until each entry of the fractal transform has been processed.

Step 6:

The main loop is incremented and if the prescribed number of iterations has not been reached, repeat back from step 3 and create a domain block in buffer B from each fractal transform entry. The process continues until the prescribed number of iterations has been reached.

At that time, the contents of buffer B are provided as data representative of the greyscale image previously compressed into the input fractal transform.

6.1.4. Implementation and Results

6.1.4.1. Examples of Fractal Images and Results

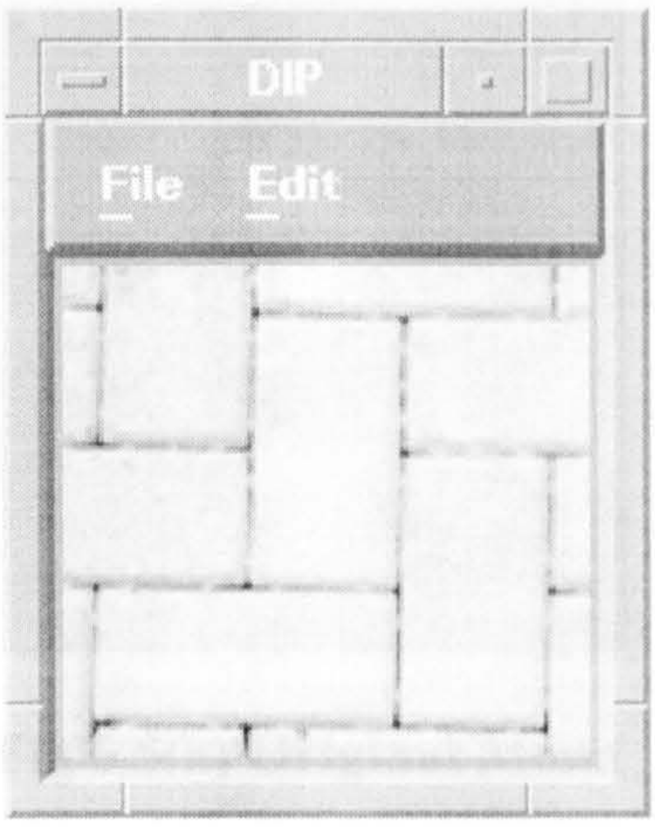


Fig 6.3(a) Original Image 1

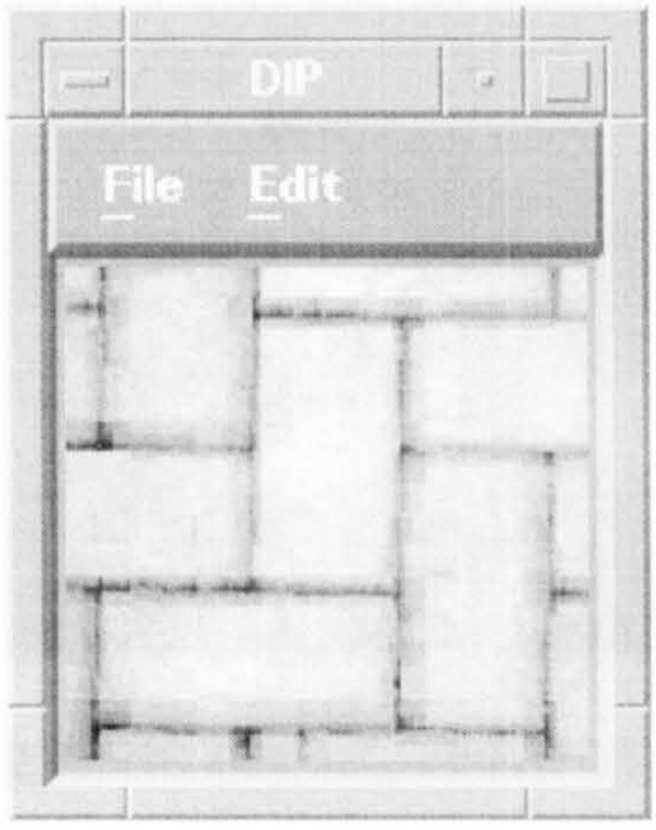


Fig 6.3(b) Decompressed Image 1

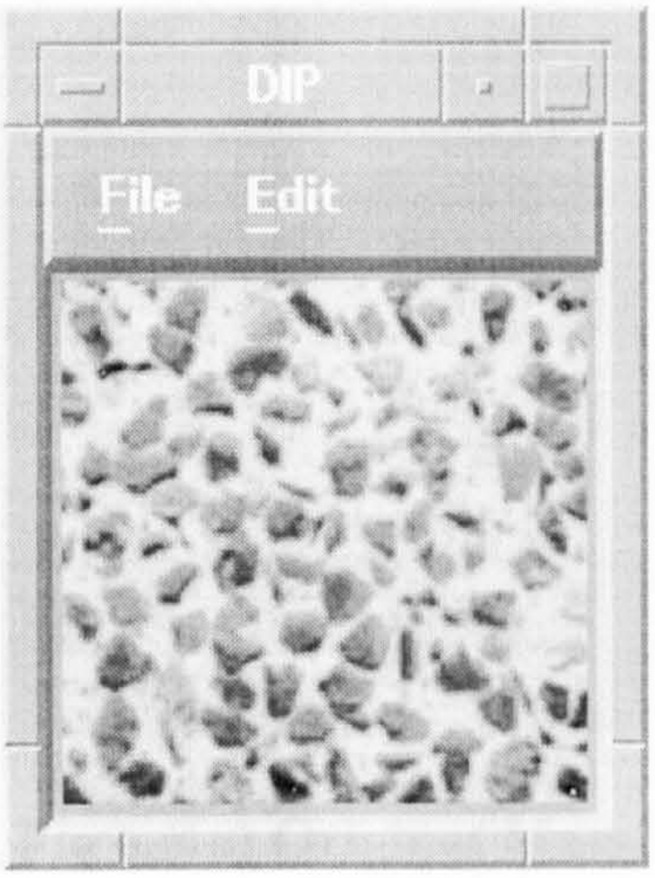


Fig 6.4(a) Original Image 2

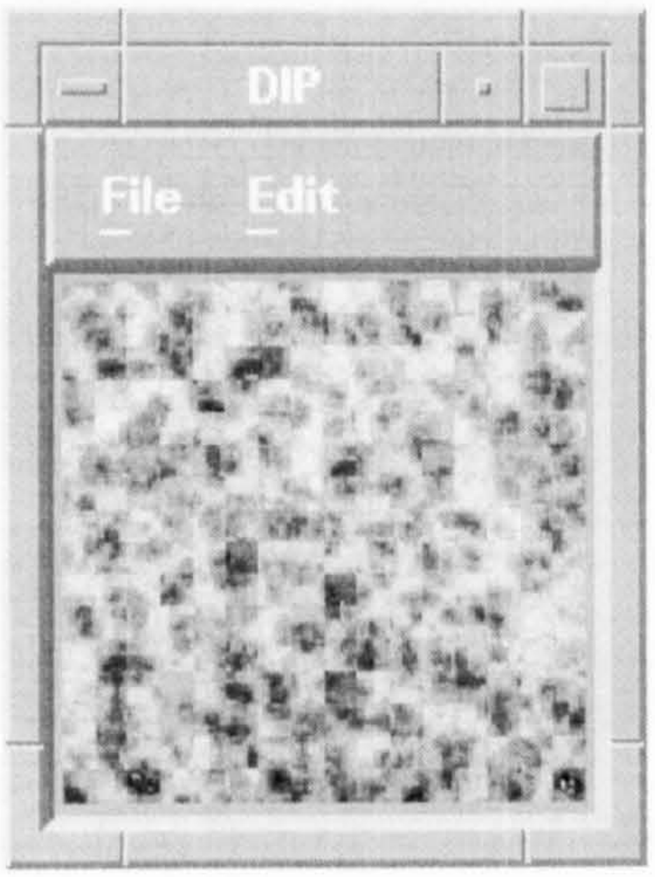


Fig 6.4(b) Decompressed Image 2

6.1.4.2. Examples of Non-Fractal Images and Results

Fig 6.5(a) Original Mona

Fig 6.5(b) Decompressed Mona

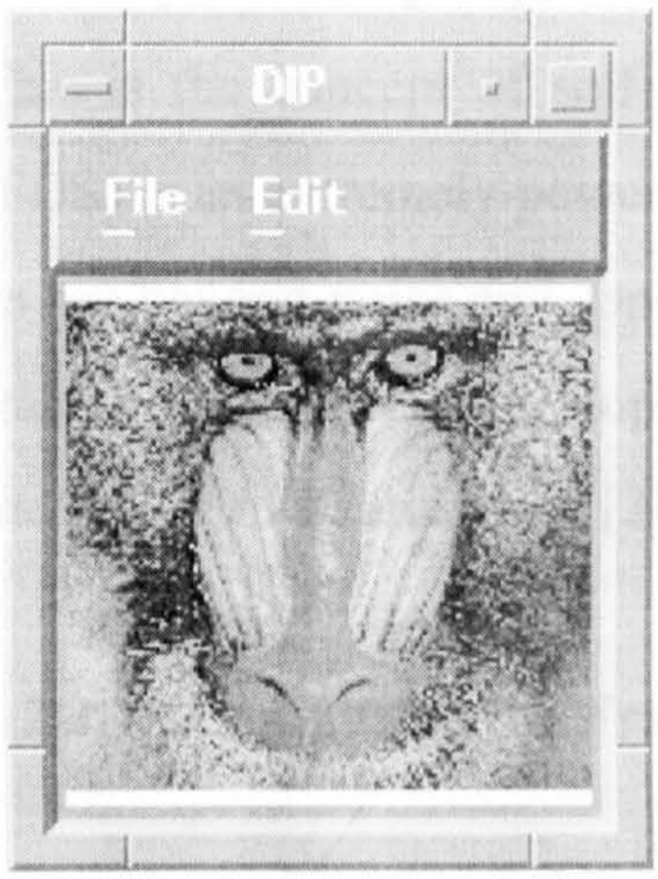


Fig 6.6(a) Original Mand

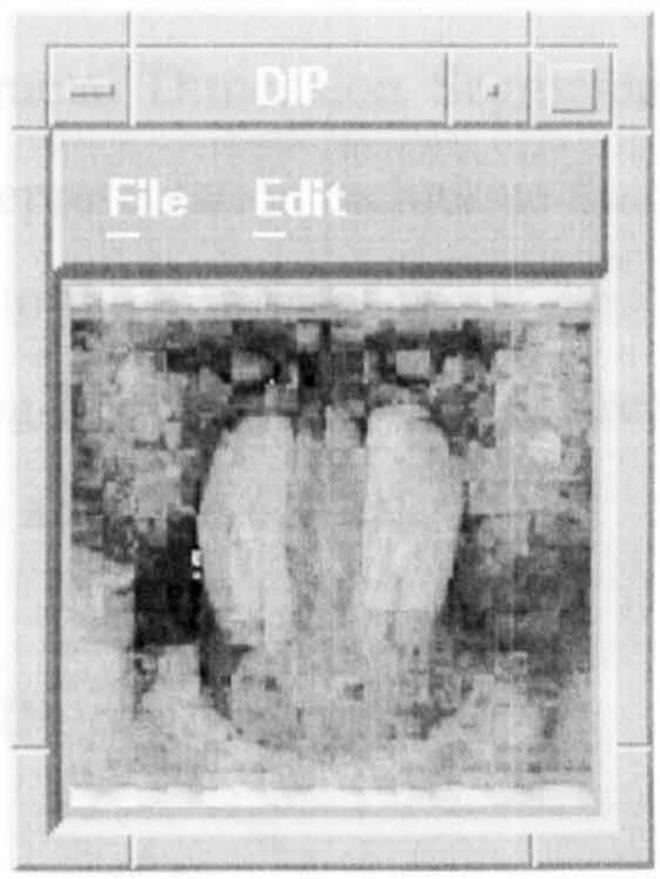


Fig 6.6(b) Decompressed Mand

6.1.5. Discussion and conclusion

The present Fractal Transform algorithm provides improved methods and apparatus for representing image data in a compressed form in a self-referential manner. The compressed data set is formed in a digital computer by a process of repeated comparisons between image data representing different small pieces of the image, known as domain blocks and shrunken range blocks, and recording the addresses of optimal matches. The image is decoded by an inverse process.

The compression ratio using this method is very high, and the quality of the reconstructed image is good, but it is not a fast method to compress an image.

6.2. Fractal Dimension Segmentation Techniques

6.2.1. Introduction

What does Fractal Dimension mean?

The fractal dimension D refers to the way in which some types of mainly naturally occurring objects in the image looks are similar at different scales. This is the concept of self-similarity [8]. Fractal Dimension Segmentation (FDS) is an extremely powerful and compact representation technique that lets us see beyond noise and can often be reduced to the history of an image. The fractal dimension is the slope of a line on a log-log plot of measured area Vs the size of measuring tool [7]. See **(Fig 6.3(a)) and (Fig 6.4(a))**.

6.2.2. Computing the Fractal Dimension (D)

Algorithms for computing fractal images are well known and are used to synthesize a variety of complex images [9]. The fractal dimension **D** provides

a measure of the roughness of a signal or the texture of an image, and to quantify the texture of an image is to compute its fractal dimension D. The number D can be computed using the standard moving window technique (**see chapter 3 section 3.7**) in which the window is moved one element at a time over an image and a set of values of D is obtained as a function of the window position. This provides visualisation and interpretation of variation in the texture characteristic of the image under the assumption that the data is statistically self-similar.

The fractal dimension D is defined as

$$\mathbf{N r^{D} = 1 \text{ and } D = \frac{\ln N}{\ln r}}$$

where

N is the number of copies for an object

r is the ratio $r < 1$

and the value of D as the fractal dimension of an image is between $2 < D < 3$.

6.2.2.1. Fractal Dimension (D)

Measuring the volume of fractals embedded into a d-dimensional Euclidean space leads to the conclusion that they are objects having *non integer* dimension. Suppose that D is the fractal dimension. There exists a smallest typical size as one cuts outs d-dimensional regions of linear L from the object and the volume. V(L) of an arbitrary object can be measured by covering it with balls of linear size L we need N(L) balls to cover it [12], so,

$$V(L) = N(L)$$

According to the modification one requires that the size of the covering balls has to diverge as well i.e. fixing L, D is defined through the scaling of N(L). The fact that an object is mathematical fractal then means that N(L) diverges as $L \rightarrow \infty$ and $l \rightarrow 0$ according to a non-integer exponent [11].

$$N(L) \approx L^{D} \qquad (1)$$

and

$$D = \lim_{L \rightarrow \infty} \frac{\ln N(L)}{\ln (l)} \qquad (2)$$

for the growing case, L = 1 therefore,

$$N(L) \approx L^{-D} \qquad (3)$$

with

$$D = \lim_{l \rightarrow 0} \frac{\ln N(l)}{\ln (1/l)} \qquad (4)$$

On the basis of (equation 4) the results for fractal dimension in 2D (two dimension or an image) is between 2 and 3 i.e.

$$2 < D < 3$$

6.2.2.2. Fourier Analysis and Fractal Dimension

A Fourier analysis of outlines of the objects enables D to be determined (Mandelbort, et, 1984). The integrated spectrum is proportional to

$$K^{-B}$$

where

$$B = 8 - 2D$$

6.2.2.3. The Histogram Features

The most basic of all image features is the measure of an image amplitude [11]. There are many degrees of freedom in establishing image amplitude features. By computing the histogram of an image we can generate a class of image features (**Fig 6.7 and Fig 6.8**). The shape of an image histogram provides many clues as to the character of the image, so that, when we are computing the fractal dimension of an image D, by any method we can compute the histogram of the image first, then implement any available method to calculate D.

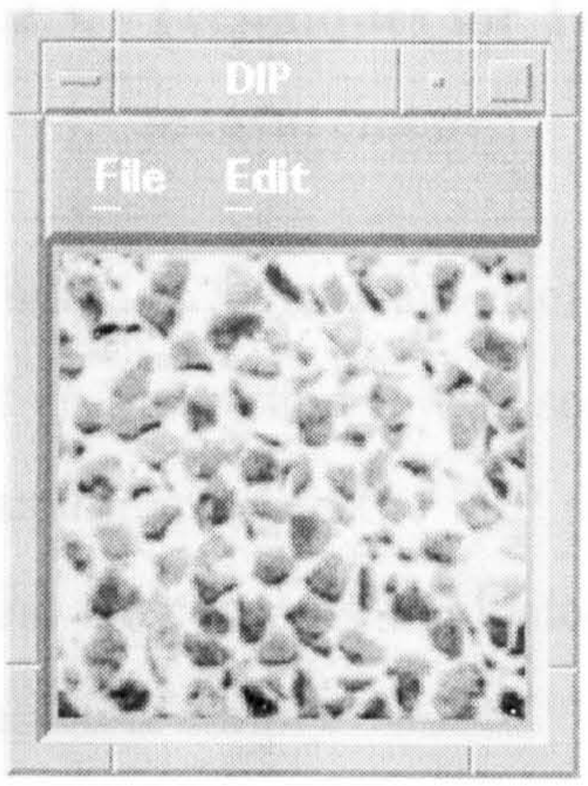


Fig 6.7(a)
Original Fractal Image3

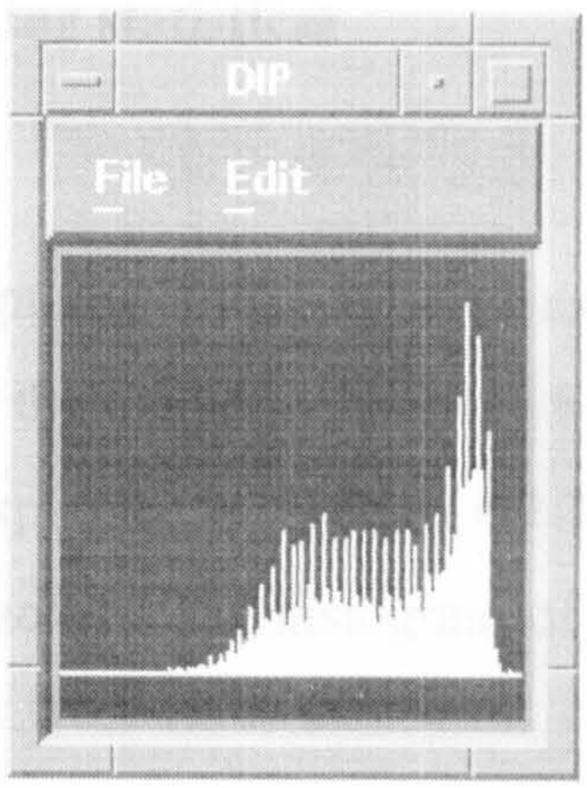


Fig 6.7(b) The Histogram

In this thesis, we are not in a position to find out which method is the best but we present the existing methods with its implementations.

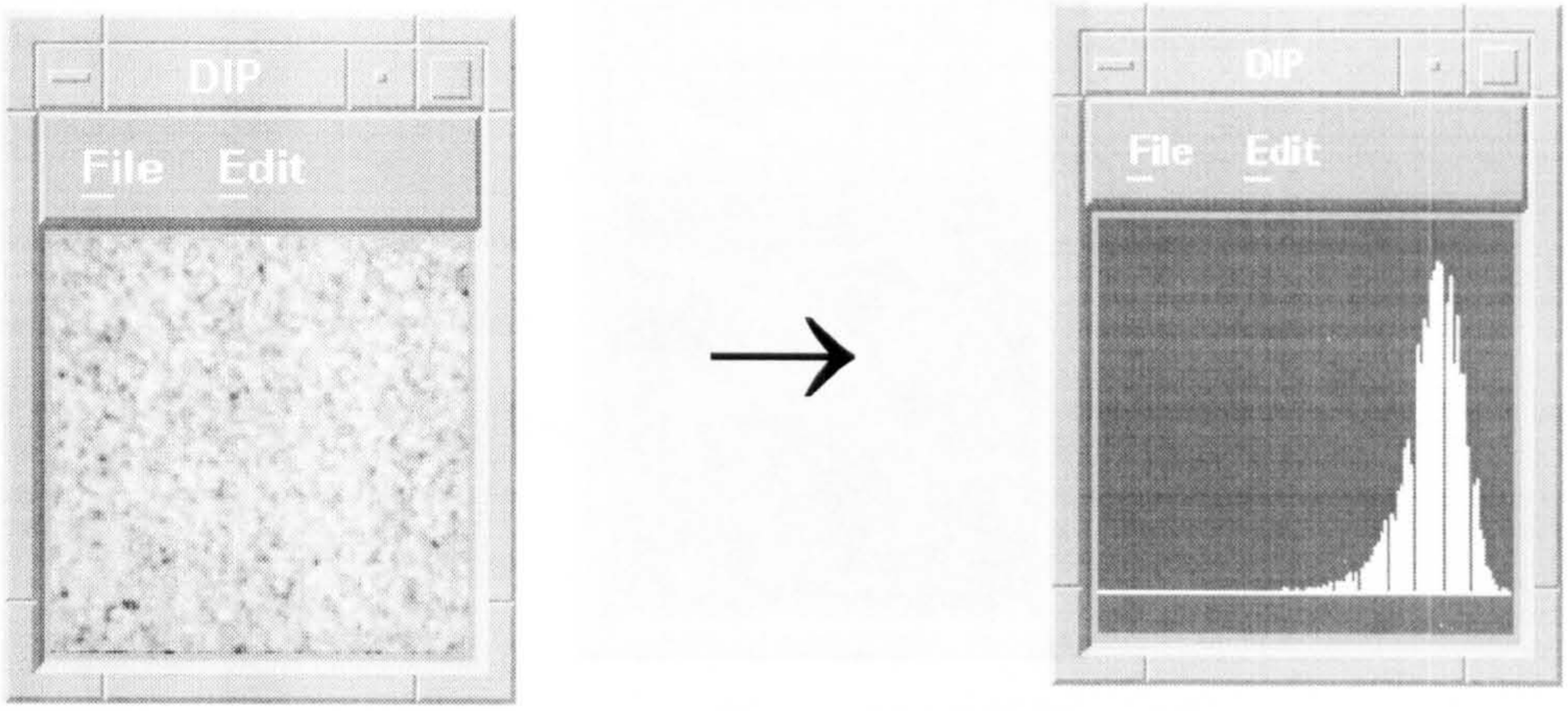


Fig 6.8(a) Original Fractal Image4

Fig 6.8(b) The Histogram

6.2.3. Techniques for Texture Segmentation using statistical measurement

There are a varieties of efficient methods for computing the fractal dimension of an image and in this section we briefly explain the methods that have been implemented in the DIP software package. As explained in **Section 6.2.2.3.**, first we compute the histogram of the image of size M. The histogram of the image is simply

$$\mathbf{H\ [b] \approx N\ (b)}$$

where

$0 \leq b \leq L - 1$

L is the histogram-maximum value

N (b) is the number of pixels of amplitude b in the window (block)

and M is the image size

The original "ship" image shown in (**Fig 6.9**) is used to test all the methods described below.

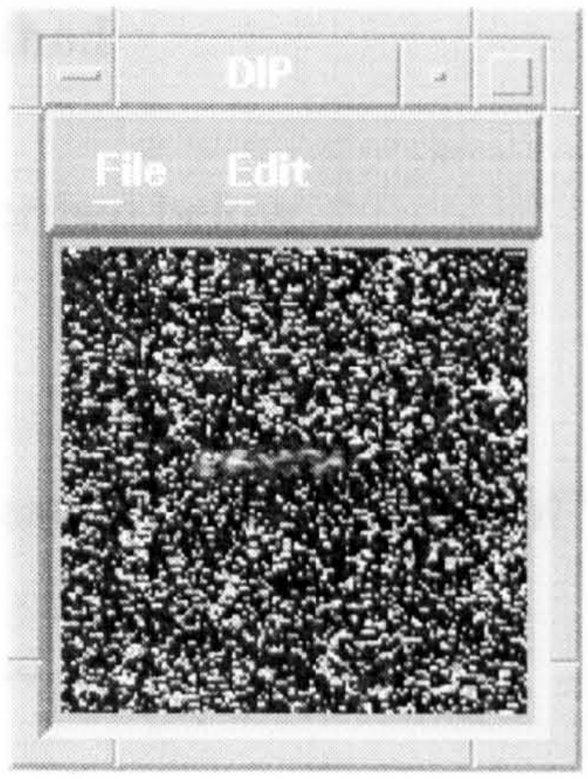


Fig 6.9 The Original "Ship" image

6.2.3.1. The Mean Method

The mean method is computed using the following equation:

$$\textbf{Mean} = \sum_{b=0}^{L-1} \left(bH[b]\right) / M^2$$

(**Fig 6.10**) shows the implementation using the mean method.

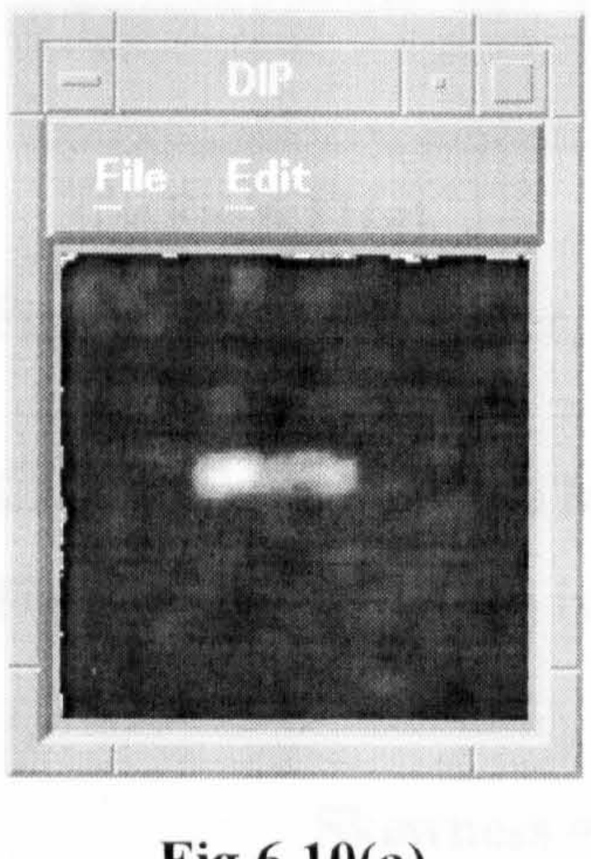


Fig 6.10(a)

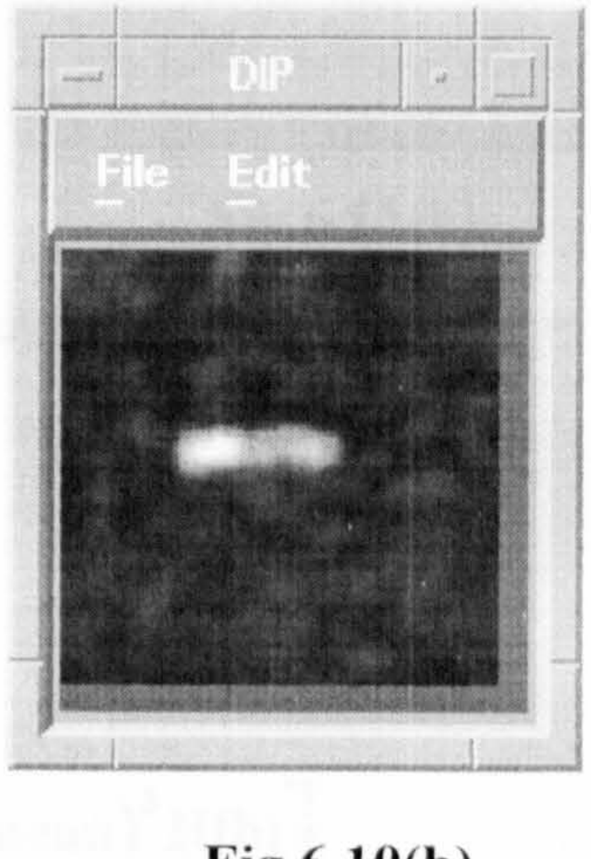


Fig 6.10(b)

Fig 6.10 (a) with border, (b) without border, window size = 8

6.2.3.2. The Variance Method

The variance method is described below

(1) Compute the mean

$$\textbf{mean} = \sum_{b=0}^{L-1} \left(bH[b]\right) / M^2$$

(2) Compute the variance

$$\textbf{variance} = \left(\sum_{b=0}^{L-1} \left((b - \textbf{mean})^2 H[b]\right) / M^2 \right)^2$$

(**Fig 6.11**) shows the implementation of the variance method.

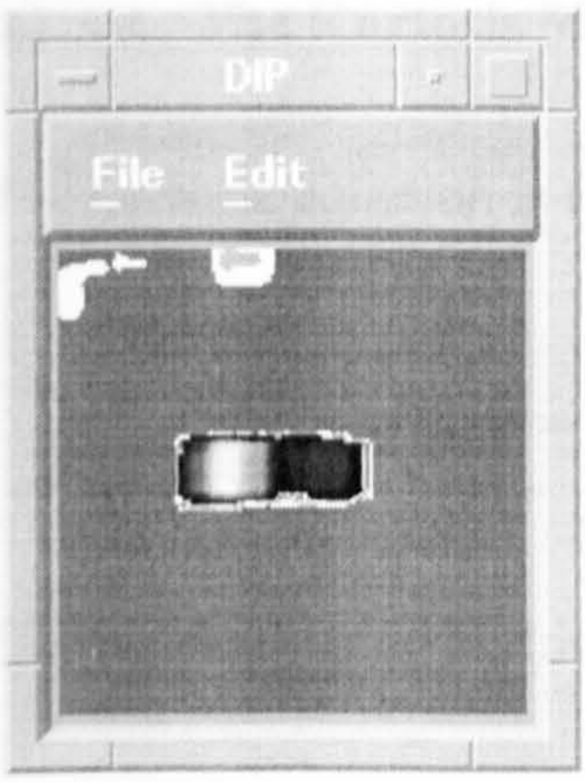


Fig 6.11(a)

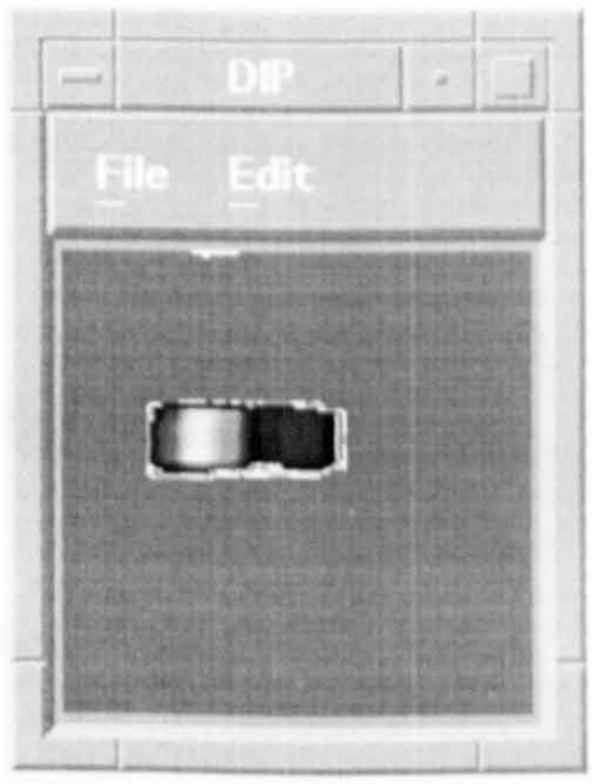


Fig 6.11(b)

Fig 6.11 (a) with border, (b) without border, window size = 16

6.2.3.3. The Skewness Method

The Skewness equation is described below:

$$\textbf{Skewness} = \frac{1}{(\textbf{variance})^3} \sum_{b=0}^{L-1} \left[(b - \textbf{mean})^3 H[b]\right]$$

The implementation of Skewness method is shown in (**Fig 6.12**).

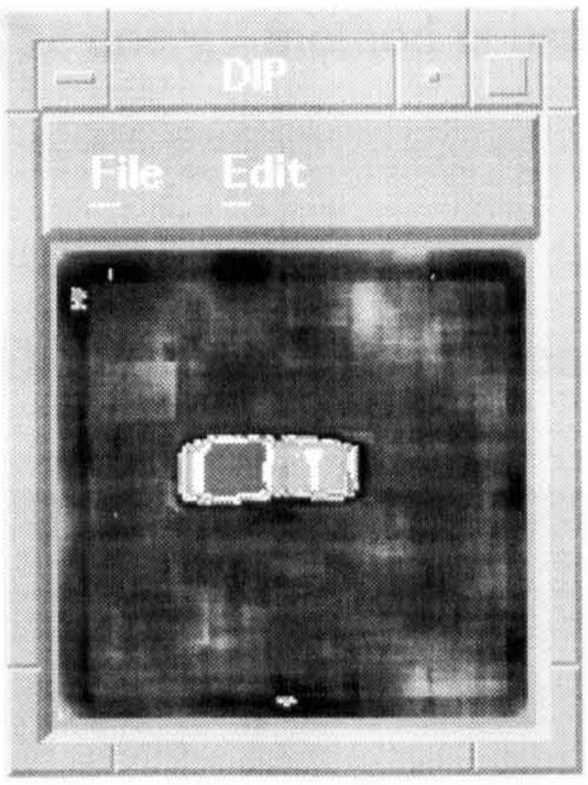


Fig 6.12(a)

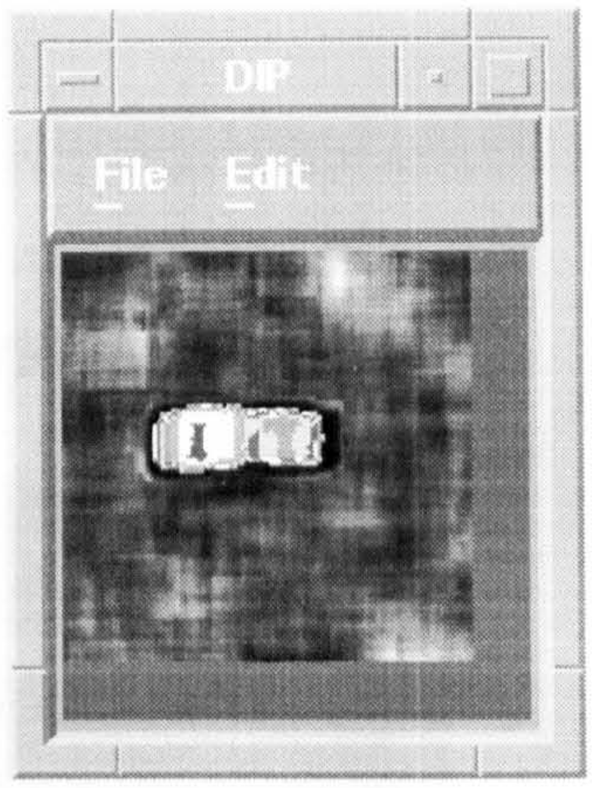


Fig 6.12(b)

Fig 6.12 (a) with border, (b) without border, window size = 16

6.2.3.4. The Kurtosis Method

The kurtosis equation is described below:

$$\textbf{Kurtosis} = \frac{1}{(\textbf{variance})^4} \sum_{b=0}^{L-1} \Big[(\textbf{b - mean})^4 \, \textbf{H[b]} - 3$$

The result of using kurtosis method is shown in (**Fig 6.13**).

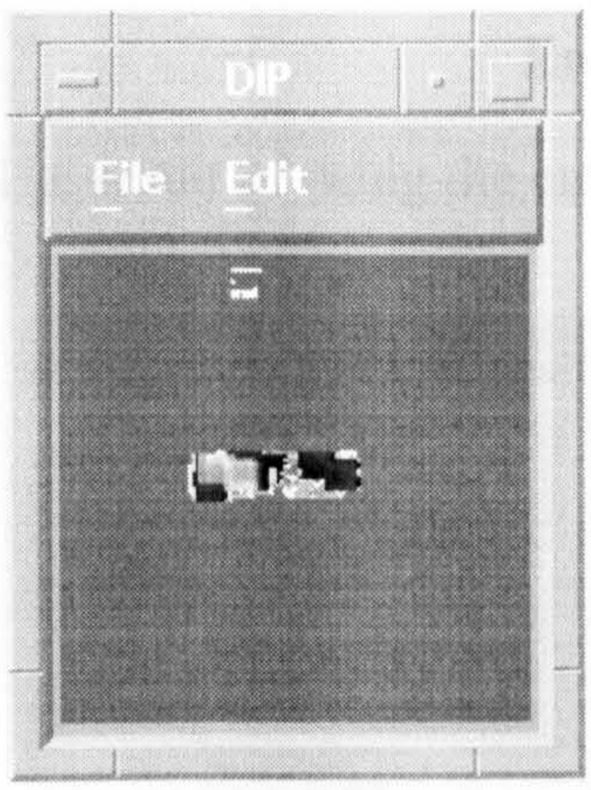


Fig 6.13(a)

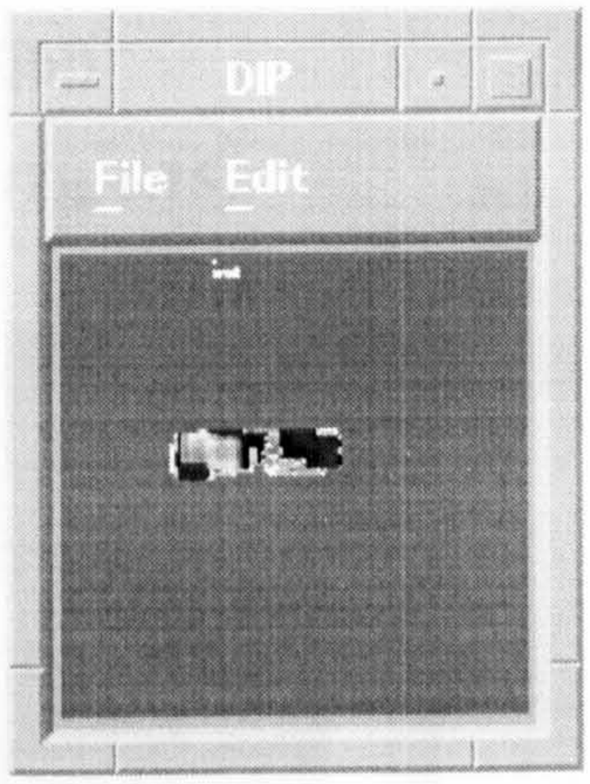


Fig 6.13(b)

Fig 6.13 (a) with border, (b) without border, window size = 8

6.2.3.5. The Energy Method

The energy method is used the following equation.

$$\textbf{Energy} = \sum_{b=0}^{L-1} \left[\mathbf{H(b)}\right]^2$$

The result of using the energy method is shown in (**Fig 6.14**).

Fig 6.14(a)

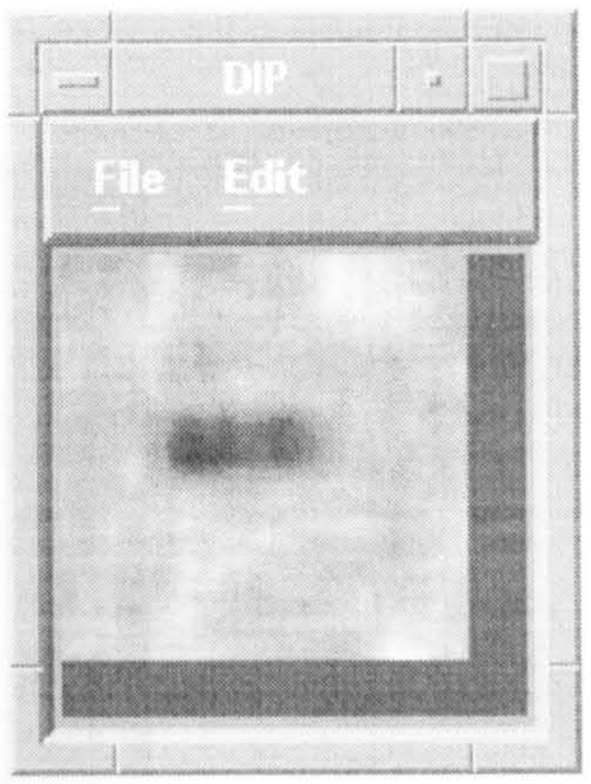


Fig 6.14(b)

Fig 6.14 : (a) with border, (b) without border, window size = 16

6.2.3.6. The Entropy Method

The entropy method is used the equation below

$$\textbf{Entropy} = -\sum_{b=0}^{L-1} \mathbf{H[b]}\log_2(\mathbf{H[b]})$$

The result of using the entropy method is shown in (**Fig 6.15**).

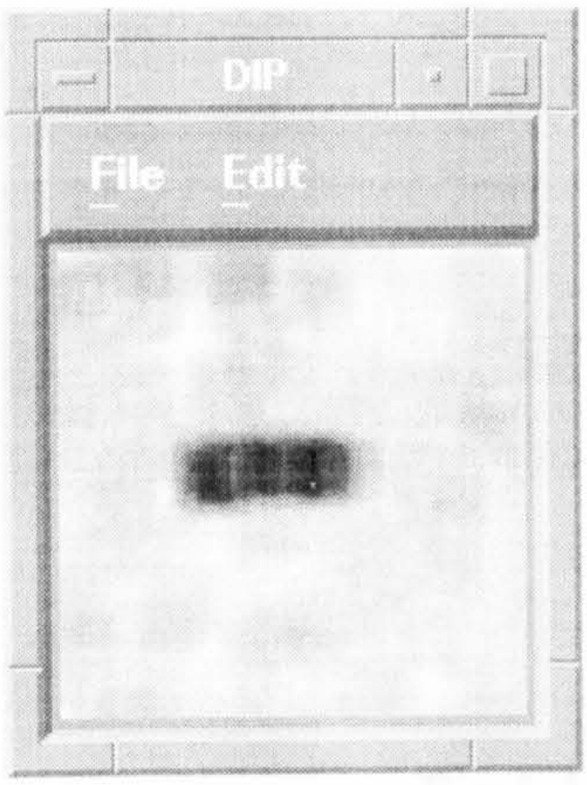


Fig 6.15(a)

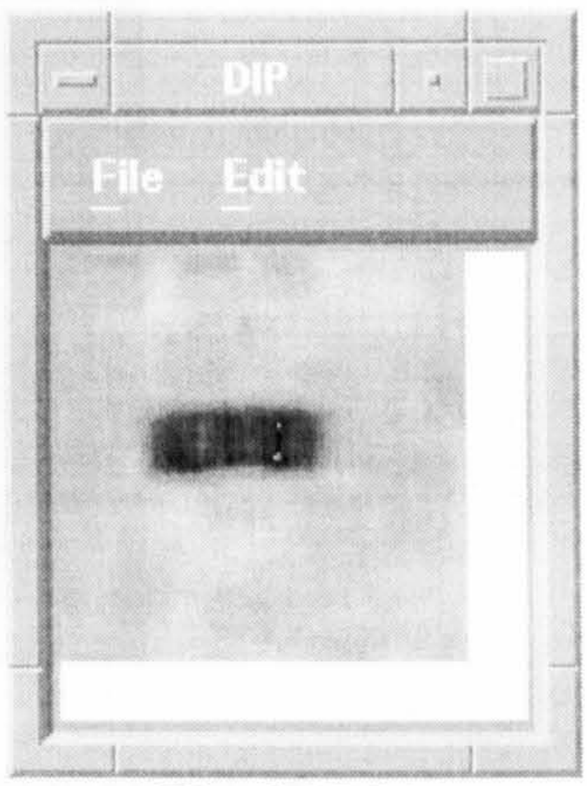


Fig 6.15(b)

Fig 6.15 : (a) with border, (b) without border, window size = 16

6.2.3.7. The Mean-diff Method

Using the mean-diff method, first we must compute the histogram of the image, then fill it up with grey scale differences between the centre pixel, and the one at the edge of the window. The mean-diff is computed in the same way as the mean method. The implementation result using the mean-diff method is shown in (**Fig 6.16**).

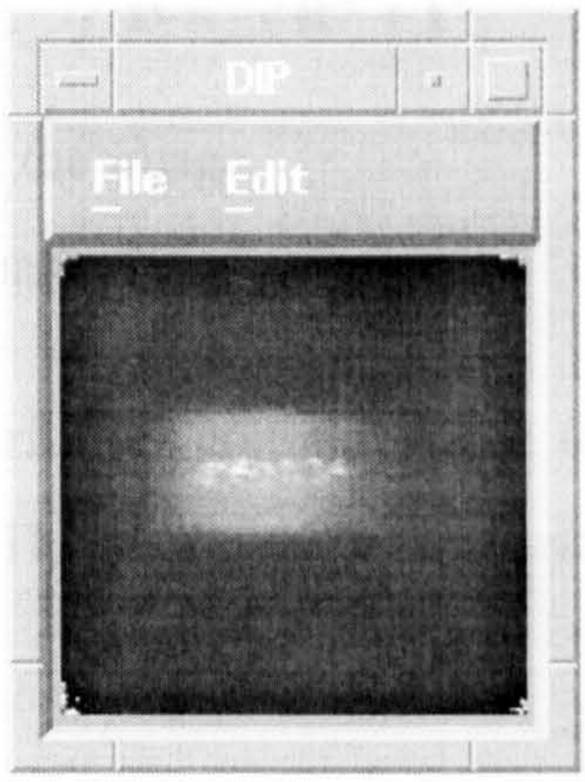


Fig 6.16

Fig 6.16 with border, window size = 32

6.2.3.8. The Variance-diff Method

The variance-diff is computed as above except that, using the variance method. The results using the variance-diff are shown in (**Fig 6.17**).

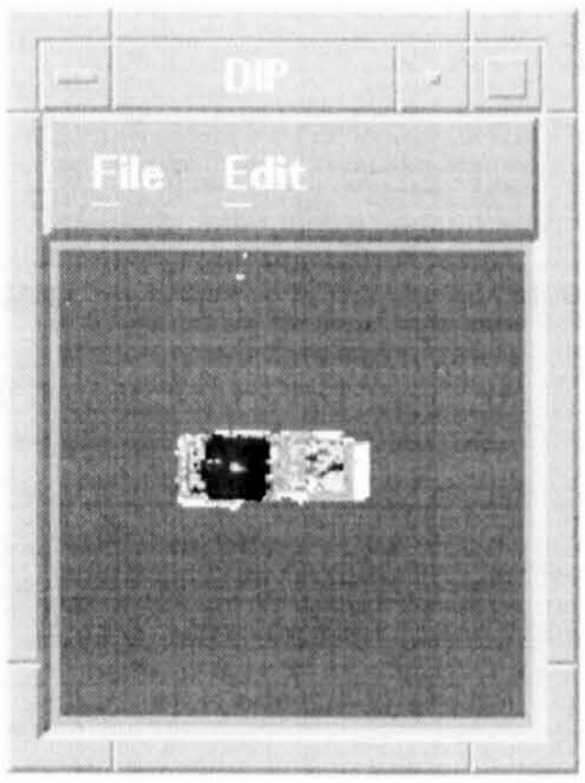


Fig 6.17

Fig 6.17 with border, window size = 16

6.2.4 Texture Segmentation using Fractal Dimension

6.2.4.1. The Power Spectrum Method

The power spectrum method is used to compute the fractal dimension D. To calculate the power spectrum, first we have to compute the Fast Fourier Transform (FFT) to get the Real and Imaginary parts of the image, then compute the D using the following equation.

$$\mathbf{D} = \sqrt{\mathbf{R}^2 + \mathbf{I}^2}$$

where **R** is the real part of the image

I is the imaginary part of the image

This method is very powerful for computing the fractal dimension, and takes extra CPU time to finish. The result using this method is shown in (**Fig 6.18**).

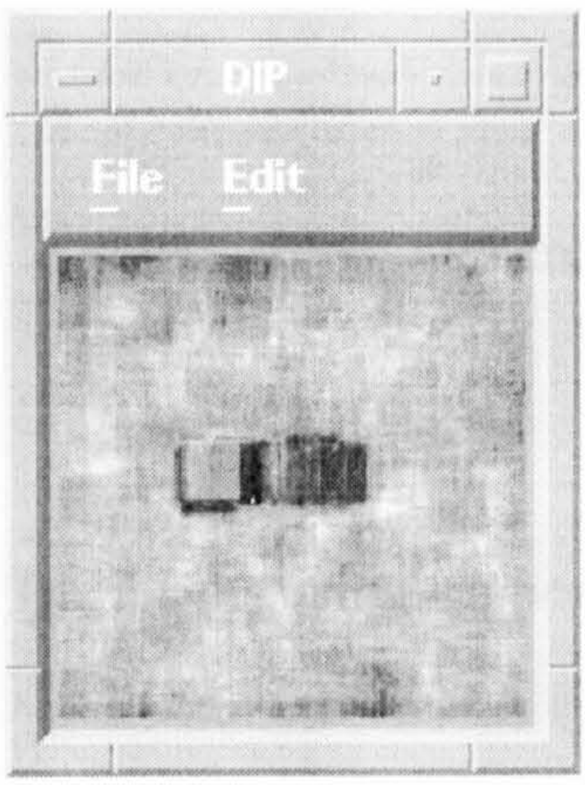


Fig 6.18(a)

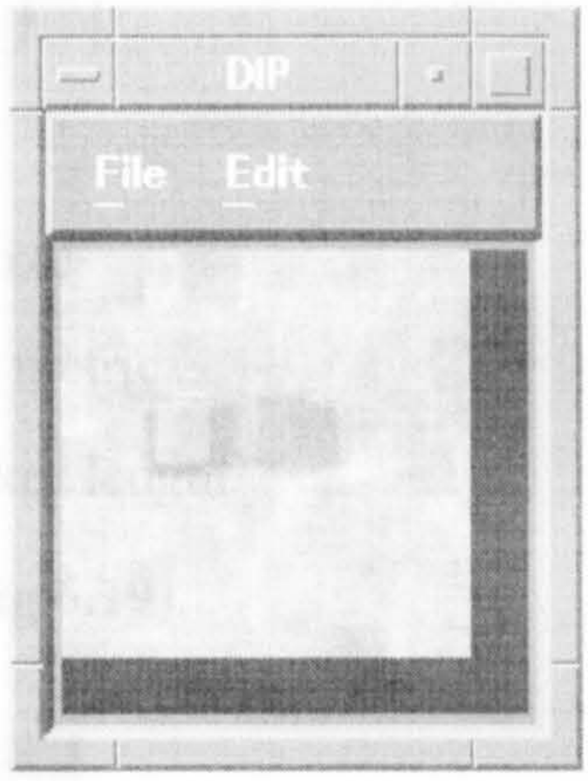


Fig 6.18(b)

Fig 6.18 D= Power Spectrum, (a) with border, (b) without border, window window size = 16

6.2.4.2. The Prism Method

One of the most powerful method for computing the fractal dimension is the prism method. This method basically splits the image up into a number of sub_squares. The central pixel is calculated as the mean of the four corners. The area of the four triangles is then calculated, and the process repeated for the next square. Having covered the entire image, the square size is halved and the process repeated. If a log-log plot is constructed for square size against total area, this gives a curve. If a straight line is drawn through the curve the fractal dimension can be found from this line. For exact fractal surfaces, this curve is a straight line. Mathematically we can describe the prism method as follows:

$$\mathbf{D = 2.0 - \lim_{n\to\infty}\left\{\frac{\ln\left(N_n(A)\right)}{\ln\left(2^n\right)}\right\}}$$

where $N_n(A)$ are number of boxes

$A \in H(R^m)$ Euclidean metric

$\left(\frac{1}{2^n}\right)$ are the squares boxes length

The results using the prism method is shown in (**Fig 6.19**).

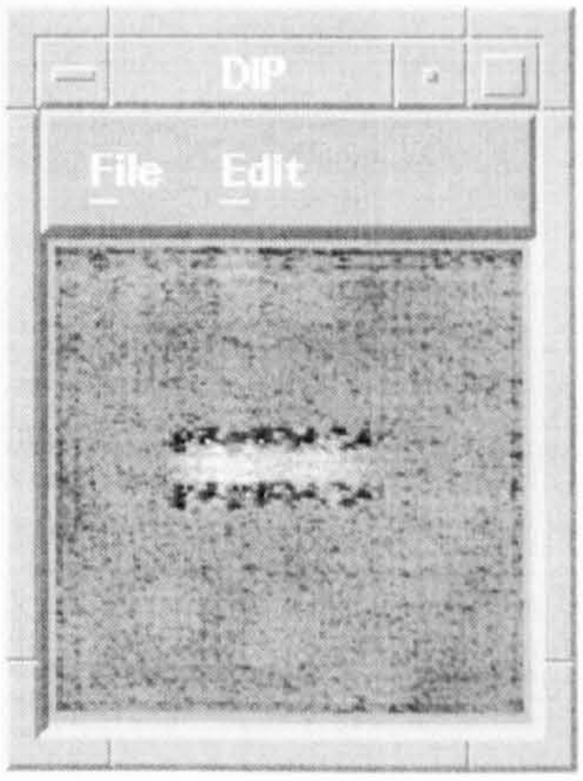


Fig 6.19 D = Prism Method, with border, window size = 16

6.2.4.3. The Box-counting Method

The Box counting method is computed the same way as the prism method, but using the following equation

$$\mathbf{D} = \lim_{n\to\infty}\left\{\frac{\ln\left(N_n(A)\right)}{\ln\left(2^n\right)}\right\}$$

where $N_n(A)$ are number of boxes

$A \in H\left(R^m\right)$ Euclidean metric

$\left(1/2^n\right)$ are the squares boxes length

The results of using the box-counting method are shown in (**Fig 6.20**).

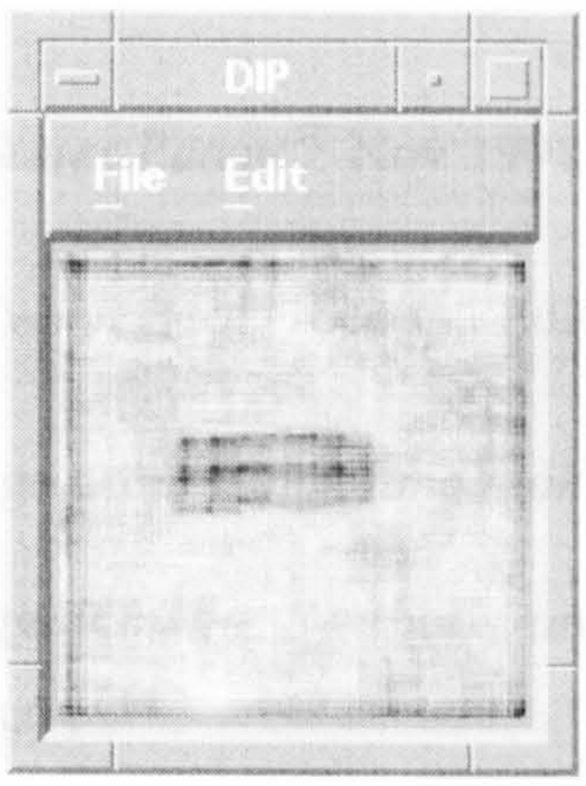


Fig 6.20 D = Box-Counting, with border, window size = 16

References

[1] Yuval Fisher, *Fractal Image Compression, Theory and Application,* 1995.

[2] Barnsley M, *Fractal Image Compression*, 1993.

[3] Arnaud E Jacquin, *Image Coding Based on Fractal Theory of Iterated Contractive Image Transformations*, IEEE Transaction on Image Processing, January 1992, Volume 1, No 1.

[4] Jon Waite, *A Review of Iterated Function System Theory for Image Compression*, 1990, British Telecom Research Laboratories, Martlesham Heath.

[5] Barnsley et al, *"United States Patent"*, Nov 1991, #5,065,447.

[6] Barnsley M, *"Fractals Everywhere"*, Second Edition, 1993.

[7] John C. Russ, *"The image processing Handbook*", Second Editon, 1994

[8] Jonathan M. Blackledge,*"On the Synthesis and processing of Fractal Signals and Images"*,IEEE 1993

[9] Peitgen,H-o., and Saup,D,Eds,*"The Science of Fractal Images"*, NewYork, Springer _Verlag,1988

[10] William K. Pratt, *"Digital Image Processing"*, 1978

[11] Tomas Vicsek,*"Fractal Growth Phenomena"* , World Scientific, 1989

[12] A.-L,Barabasi and H.E. Stanley,*"Fractal Concepts in Surface Growth".*

[13] G.Cherbit, *"Fractals non-intergeral Dimensions and Applications"*, 1990

[14] J.M Beaumont, "*Image Data Compression Using Fractal Techniques*", BT Technology Journal 9, 93 - 109 (1991).

[15] M.Barnsley, *Fractal Everywhere*, 1988.

[16] A. Jacquin, *Image Coding Based on a Fractal Theory of Iterated Contractive Image* Transform, IEEE Transactions on Image Processing I, 18 - 30 (1992).

[17] M. Barnsley and A. Sloan, *"Method and Apparatus for Processing Digital Data"*, US Patent #5,065,447.

[18] J. Waite, *"A Review of Iterated Function System Theory for Image Compression"*, preprint, BT Laboratories, Martlesham Heath, 1992.

Chapter 7

Pattern Recognition under Fractal Transform Image Compression

This chapter defines the problems with pattern recognition, explains two pattern recognition methods generally used and provide details for the solution of these problems.

In this chapter **Page No**

7.1. Problems with Recognition

Pattern recognition is increasingly being used in image processing for many fields such as handwriting, character, speech, remote sensing, motion detection recognition, etc.

There are many techniques used for automatic pattern recognition [1]. Artificial Neural Networks (ANN) is the most widely used technique for pattern recognition. Template matching, correlation or matching filtering are the methods used in this chapter.

These methods fail to recognise a pattern if it has changed its orientation in the plane or scale .

In this chapter, we provide a solution to these problems by using a Fractal Image Compression technique. We applied generally used pattern recognition methods, which are (i) Cross-Correlation method and (ii) Least Squares method for comparison between these methods and the new solution.

7.2. Conventional Template Matching Techniques

Here we provide greater details associated with each method we have used and its implementations.

7.2.1. Cross-Correlation Method

The Cross-Correlation method is derived from the mean square error equation. If we assume that we have an input pattern f_{ij}, and an identically dimensioned template pattern p_{ij}, the matching (recognition) is obtained when the error is minimum.

Consider

$$\mathbf{e} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left(\mathbf{f}_{i,j} - \mathbf{p}_{i,j}\right)^2$$

Expanding this equation , we get

$$\mathbf{e} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left(\mathbf{f}^2_{i+k,j+l} - 2\mathbf{f}_{i+k,j+l}\mathbf{p}_{i+k,j+l} + \mathbf{p}^2_{i+k,j+l}\right)$$

$$= \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\mathbf{f}^2_{i+k,j+l} - 2\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\mathbf{f}_{i+k,j+l}\mathbf{p}_{i+k,j+l} + \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\mathbf{p}^2_{i+k,j+l}$$

if we now assume that $\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\mathbf{f}^2_{i+k,j+l}$ and $\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\mathbf{p}^2_{i+k,j+l}$ are regular constants.

then the error is $\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\mathbf{f}_{i+k,j+l}\cdot\mathbf{p}_{i+k,j+l}$ which is that a ***Cross-Correlation*** between the pattern $\mathbf{f}_{ij}$ and the template pattern $\mathbf{p}_{ij}$. In addition we can filter out the effects of quantization and threshold noise before template matching is performed. Implementations and results using this method are illustrated in (**Fig 7.1** and **Fig 7.2**) where the square in the image on the left indicates where the correlation surface is a maximum.

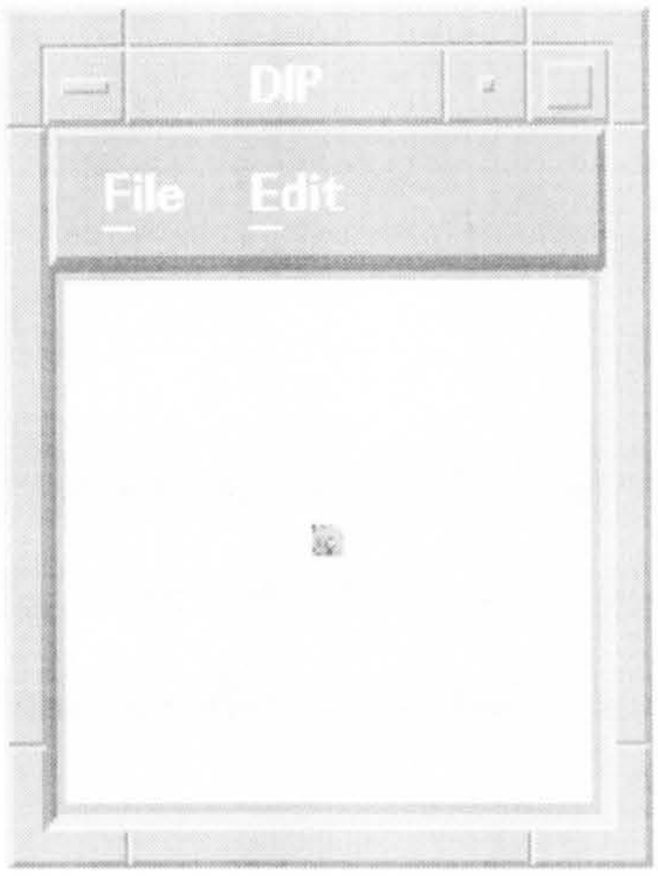


Fig 7.1
Object Pattern(size 8x8)

Fig 7.1(a)
Pattern Recognition

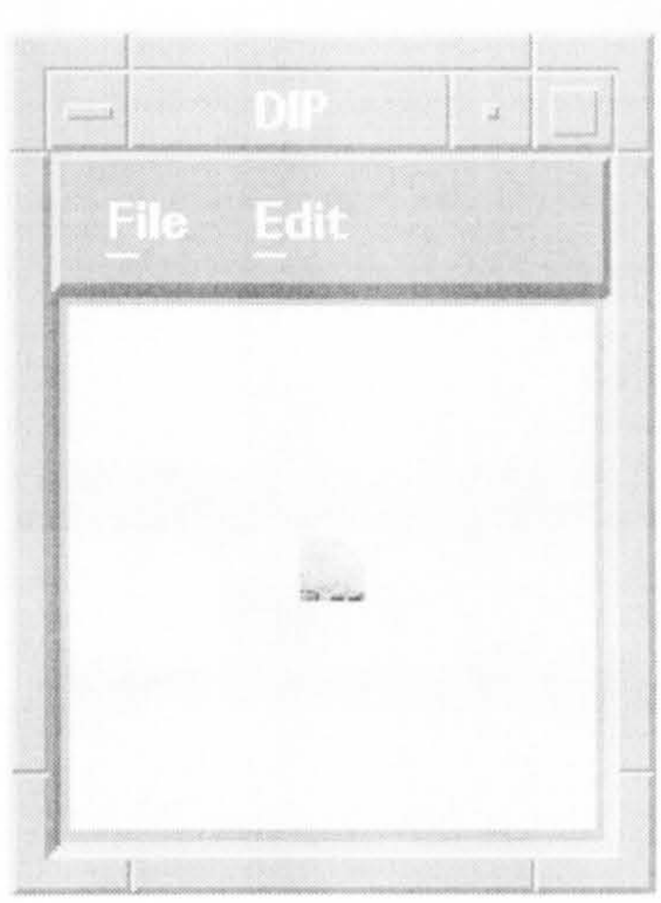


Fig 7.2
Object Pattern(size 16x16)

Fig 7.2(a)
Pattern Recognition

7.2.2. Least-Squares Method

Least-squares matching is perhaps the simplest pattern recognition technique. The patterns are identified by comparing the input pattern to a list of stored

pattern representations (templates). The principle is based on finding the minimum of the error function.

$$\mathbf{e} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left(\mathbf{f}_{i,j} - \mathbf{p}_{i,j} \right)^2 \qquad (2)$$

Implementations and results using this approach are illustrated in (**Fig 7.3** and **Fig 7.4**) with different size of patterns where again the square in the image indicates the minimum of **e**.

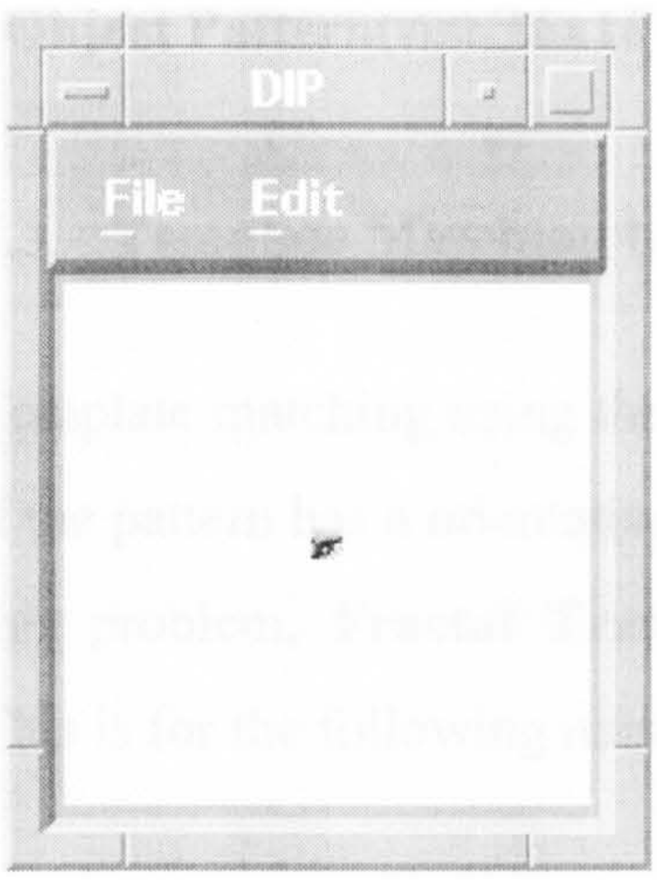


Fig 7.3
Object Pattern(size 8x8)

Fig 7.3(a)
Pattern Recognition

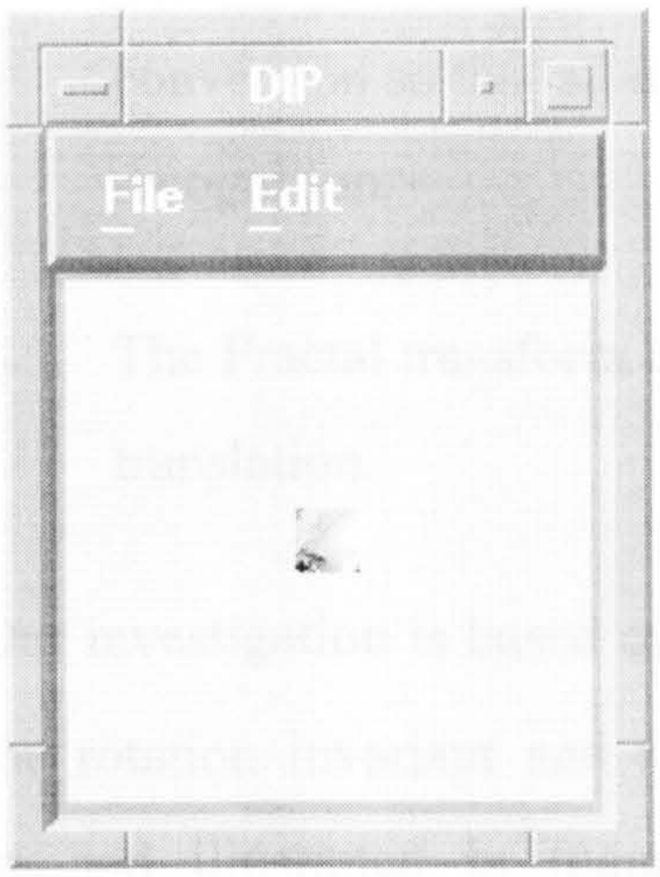


Fig 7.4
Object Pattern(size 16x16)

Fig 7.4(a)
Pattern Recognition

7.3. Template Matching Using Fractal Image Compression Technique

Template matching using the square or correlation method are not appropriate if the pattern has a orientation (rotated) or is a different size (scaled). To solve this problem, **Fractal Transform Image Compression** has been applied. This is for the following reasons:

1. By compressing the image we work with a smaller size of data image hence we speeds up the recognition operation.

2. The Fractal Transform Image Compression produces a readable compressed file unlike other compressors for example JPEG Compressor or Differential Image Compressor[5].

3. The Compressed data is easy to analyse by mathematical equations.

4. The compressed data could be used as input layers to ANN with small conversion so that all data should be between 0 and 1 for pattern recognition.

5. The Fractal transform copes with transformation by rotation, scaling and translation.

Our investigation is based on (5) and this thesis concentrates on how to solve the rotation invariant and some scaling problems using the L^1 distance [4] method illustrated in (equation **3**) used by the fractal transform image compression.

7.3.1. Automatic Pattern Recognition Algorithm

The automatic pattern recognition algorithm designed using fractal transform compression is as follows.

Step 1: Clip an object (pattern) from the image.

Step 2: Compress the pattern using Fractal Transform Compression technique.

Step 3: Write the compressed pattern data into a file.

Step 4: Move a window using the standard moving window method (template pattern) of identical dimension as the object pattern, around the image, pixel by pixel, from top to with Fractal compression technique.

Step 5: Write compressed template pattern data into another file.

Step 6: Repeat step 4 and step 5 until the window (template pattern)

has covered the whole image.

Step 7: The L^1 distance method is used to find the matching input pattern P on the templates pattern F compressed data file. The L^1 method is used because it can be calculated faster and easier than the Hausdorff distance[3][4] L^1 and can be described as follows:

$$L^1 = \left| \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left(f_{i,j} - p_{i,j} \right) \right| \qquad (3)$$

and if L^1 is small (i.e. $L^1 = 0$) then the two patterns (images) are alike.

7.3.2. Solution to the Rotation invariant Problem

The first and major problem for pattern recognition (i.e. rotation invariant) has been solved.

We note that (*when we compress the input pattern and then rotate the input pattern by 0° or 90° or 180° or 270° or 360° then compress the rotated input pattern*) the compressed files for both input pattern and rotated pattern are **"different"** but the **"same"** under the L^1 matching method, here the word "same" means matching or the error is minimum; and also with various rotation degrees. In appendix E shows all the implementation results using the **APR** "**A**utomatic **P**attern **R**ecognition" method, on black/white images and greyscales images. The figures (**Fig 7.5** to **Fig 7.13**) shows the results of implementing the above algorithm on Black and White images, and the figures (**Fig 17.14** to **Fig 7.33**) shows the results of implementing the above algorithm on greyscales images. The figures (**Fig 8.1 to Fig 8.25**) shows the original greyscales images but in 3D space view, the Cross-Correlation with a single peak, the miss match "no correlation" (i.e. no peak), and the recognition using

the **APR** "**A**utomatic **P**attern **R**ecognition" method respectively, with one or more than one peak, depend on if there are more than one similar object in the image to the located object (input pattern), even with variant rotation angle.

7.3.3. Solution to the Scaling Problem

Thus, the Fractal Transform Compression does the scaling or shrinking only by power of 2 [2]. We notice that (when we compress the input pattern and then scale the object in the input pattern (i.e. zooming in or out), the object must stay inside the pattern window if we zoom out). The compressed file for both the input pattern, and the scaling object pattern are "**different**" but the "**same**" when we apply the matching equation as following:

$$\mathbf{e} = \left| \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left(\mathbf{f}_{i,j} - \mathbf{p}_{i,j} \right) \right|$$

The implementations and results are shown in appendix E (**Fig 7.34**) and (**Fig 7.35**)

NB For the result purpose, we used the standard 8x8 pattern (JPEG Committee) for the size of the input and template patterns.

References

[1] Don Pearson, *"Image Processing"*, 1991.

[2] M.Barnsley, *"Fractal Everywhere"*, Second Edition, 1993.

[3] Yuval Fisher, *"Fractal Image Compression, Theory and Applications"*, 1995.

[4] M. Barnsley and A. Sloan, *"Method and Apparatus for Processing Digital Data"*, US Patent #5,065,447.

Chapter 8

Conclusion

The ***Fractal Transform Image compression*** technique developed for this thesis has solved the rotation and scale variant pattern recognition problem. This method has been compared with different image compression algorithms such as JPEG, Block-Trunction-Coding, Differential Compression and Pixel-Average Compression and pattern recognition methods such as Least Squares method and Cross_Correlation method. The approach used for solving the pattern recognition problem has been fully explained and implemented. This solution opens the door to many real world pattern recognition problems to be reconsidered such as fingerprint recognition, character recognition, signature recognition, medical image recognition (X-ray, MRI, microscope images, remote sensing, etc.). The **automatic** pattern recognition solution has been successfully tested on different images with various rotation angles. Neither the reference image nor the input pattern require any pre-process for extracting features. The Automatic Pattern Recognition method is performed for both compressed input pattern and reference image and the $\mathbf{L^1}$ distance method is used because it can be calculated faster and easier than the Hausdorff distance. This makes the new Automatic Pattern Recognition method fast to use. **Further** work could be undertaken in this area using 3-D images so that an object (image) which has been rotated around the Z axis could be recognised by use of the fractal transform image compression technique for 3-D images giving similar results to those seen in the 2D images. A DIP Software Package has been developed as part of this thesis as a programmer workload and may be of value to future research in this area of image analysis. The DIP package is written using C language and X Designer with Motif and ANSI library. This package is running under UNIX 9.01 operating system.

Appendix A

Installing and Running DIP

This appendix describes all the steps needed for installing and running the DIP package.

In this appendix: **Page No**

The disk which is included with this thesis contains all the programmes for DIP Software Package. The following steps are used to install the DIP package from the floppy disk into the HP workstation under UNIX operating system.

Step 1: In any directory you are, create a subdirectory named DIP to contain all the files associated with the program package.

$mkdir DIP

Step 2: Get into the DIP directory by typing

$cd DIP

Step 3: Create other subdirectories under DIP and name them interfac, include and program

$mkdir algorith interfac include program

A1.1. Installing the DIP functions

Step 4: Move to algorith directory

$cd algorith

This directory will contain all the function associated with the DIP algorithms.

Step 5: To set the current working directory on the floppy disk (the floppy disk has the same directory structures as those we have just created in the Hewlett Packard workstation).

$cd /floppy/algorith

Step 6: Transfer all the files with c extension from the disk.

$mcopy -t "a:*.c"

Step 7: Get out from algorith directory

$cd ..

A1.2. Installing the Interface Functions

Step 8: Move to interfac directory.

$cd interfac

Step 9: Move to interfac directory in the floppy disk.

$cd /floppy/interfac

Step 10: Transfer all the files with c extension from the disk.

$mcopy -t "a:*.c"

A1.3. Installing the Header Files

Step 11: Move to include directory.

$cd include

Step 12: Move to include directory in the floppy disk.

$cd /floppy/include

Step 13: Transfer all the files with h extension from the disk.

$mcopy -t "a:*.h"

Step 14: Move to DIP directory.

$cd ..

$cd DIP

A1.4. Installing the Main Program

Step 15: Move to program directory.

$cd program

This directory will contain the main program, the images used by the package and the label pixmap which are loaded each time the package starts.

Step 16: To set current working directory on the floppy disk.

$cd /floppy/program

Step 17: Transfer the main program.

$mcopy "a: main.c"

Step 18: Transfer all the images.

$mcopy "a:*.xva"

Step 19: Transfer the label pixmap.

$mcopy "a: my-label.xpm"

A.2. Running the DIP Program

Before running the DIP package we must compile and create executable file.

A.2.1. Compiling the Files

Step 20: Compile the files and create an archive library using the following commands

```
$cd algorith
$cc -Aa -c *.c
$ar cr dip.a *.o
```

Step 21: Move the library dip.a to program directory by typing

```
$ mv dip.a ../program
```

Step 22: Compile the interface files and create an archive library using the following commands.

```
$cd interface
$cc -Aa -c *.c -I /usr/include/Motif1.2 -I /usr/include/X11R5
$ar cr interface.a *.o
```

Step 23: Move the interface.a library to the program directory.

```
$mv interface.a ../program
```

A2.2. Run the DIP Package

Generate the executable file DIP.

Step 24: Move to program directory which contains the main program.

```
$cd program
```

Step 25: Compile the main program by typing

$cc -Aa -c main.c -I /usr/include/Motif1.2 -I / usr/include/X11R5

Step 26: Install and run the DIP program by typing

$cc -Aa -o DIP main.o dip.a interface.a -L /usr/lip/Motif1.2 -L /usr/lip/X11R5 -lXm -lXt -lX11 -lm

Step 27: The user may now run the program by typing

$DIP

Appendix B

The X Designer Software

This appendix provides an introduction to the X-Designer Software Package that has been used to generate the interface, and work bench on which this research has been carried out.

In this appendix **Page No**

B.0. X Window System - X Designer

Since the principal system used to develop the software on which this research is based is the X-Window system, a brief overview to the X-Window system and there case tool used (X-Designer) to generated Motif code is given.

B.1. Introduction

In 1984, the Massachusetts Institute of Technology (MIT) discovered that it had a problem. It had acquired a motley collection of incompatible graphical workstations from a variety of hardware vendors. From these, it wanted to build a network of graphical workstations to be used as teaching aids. Since the workstations were incompatible, there was no easy solution.

To solve this problem, MIT (in conjunction with DEC and IBM) set up project Athena to produce a networked, vendor-independent windowing system. The project Athena workers made use of some ideas from an earlier window system from Stanford University, called **W**. With startling originality, they called the result **X**.

B.2. What X is

The X window system, or X, is a network-transparent system which has been implemented for UNIX, UMS, DOS, Mackintosh and other operating systems. With X, one can run multiple applications simultaneously in windows, generating text and graphics in monochrome or color on a bitmap display.

Network transparency means that one can use application programs that are running on other machines scattered throughout the network as if they were running on your machine. Because X permits applications to be device

independent, applications need not be rewritten, recompiled, or even relinked to work with new display hardware.

X is based on a client/server model. In this model, the application program does not access the screen or receive keyboard and mouse input directly. Instead, the application program (the client) communicates via messages with a second progress, the X server.

It is the X server which manages windows on the screen, creates text and graphics, and handles the keyboard and mouse.

The client application does not have to know about the display hardware it just sends a request message to the X server, to say, display a window, and the X server takes care of the rest.
Typically, a single server will have multiple clients - that is, the user will have many client applications visible on his screen. A single client can also use multiple servers, and so drive multiple screens and take input from many places. A client and server may also be the same machine.

The client does not have to know anything about the display or input hardware - that is handled by a hardware-specific device in the X server. Similarly, the X server does not need to know anything about the nature of the client application. All that is necessary is that the client and server agree on the set of messages they can exchange - the protocol.
It is the X protocol which is the basis of the X window system.

The X protocol is low-level. Typical messages from clients to server are of the form “display a window here”, or “draw a line from here to there”. The client will also receive messages from the server, such as “mouse button 1 pressed in this window”. The fact that the protocol is low-level simply means

that it is easy to standardise. However, it means that writing applications which use X directly is a slow and frustrating business, and there is no guarantee that two applications written to use X will look or behave anything like each other.

B.3. X Terminals, Workstations and PC's

The early X servers were implemented as software running on networked workstations. Once the protocol was stabilised, hardware manufacturers were quick to produce dedicated X terminals, which implement all or part of the X server in hardware.

We do not have to have an X terminal to use X. We can get X server software for almost any workstation, as well as IBM PC's and Mackintoshes. In addition, we can emulate many standard character-based terminals in an X window, so we can use X servers to access existing applications. Whatever our current mix of hardware, we can use X to create a network of standard graphical terminals, and access our applications from any terminal.

B.4. The Structure of X

Before developing an application based on X, it is as well to know a little bit about the internal structure of X. There are a number of layers at the bottom of Xlib. This is a C language library which communicates across the network to an X server. It is the component which insulates the application programmer from the details of the protocol. Application programs can be written using just the facilities of Xlib, but rarely are.

The X Toolkit (Xt) intrinsics sit on top of Xlib. They provide a set of facilities used for building and manipulating 'widgets'. However application

programmers do not normally build widgets themselves - they make use of predefined widget sets, of which probably the most widely supported is the OFS/Motif widget set.

Above the Xt intrinsics sits the widget set. **Widgets** correspond to abstract user interface elements such as buttons, sliders, menus and text fields. The widget set is not part of the X window system, it is a tool to make the benefits of X (primarily networking and vendor independence) accessible to an average application programmer.

B.5. OSF/Motif Toolkit

The Xlib library and the X protocol provide windowing functions at a very low level of abstraction - too low for application programmers, and too low to guarantee consistency between applications. To provide a consistent look and feel, and to make application building practical, we need widgets. Widgets are abstract components used by an application programmer. They include things like buttons, scrollbars, menus, dialog boxes and many others. As well as using the application programmer's task, widgets aid consistency - each widget has a specified appearance and behaviour, so applications built from a given set of widgets will have a similar look and feel.

The Motif toolkit which was defined by the Open Software Foundation (OSF), gives all the widgets the user would expect to find in a GUI-builder's kit. In combination, these widgets let the user create graphical user interfaces for an applications. PC users may realise that the SOF/Motif toolkit looks and feels like Microsoft Windows.

B.6. Basic X Toolkit Terminology and Concepts

The Motif user-interface specification is completely independent of how it is implemented. In other words, we do not have to use the X window system to implement a Motif style graphical user interface (GUI). However, to enhance portability and robustness, the OSF chose to implement the Motif GUI using X as the window system and the X Toolkit Intrinsics as the platform for the Application Programmer's Interface (API).

Xt provides an object-oriented framework for creating reusable, configurable user-interface components called widgets. Motif provides widgets for such common user-interface elements as labels, buttons, menus, dialog boxes, scrollbars, and text-entry or display areas. In addition, there are widgets called managers, whose only job is to control the layout of other widgets, so the application doesn't have to worry about details of widget placement when the application is moved or resized.

A widget operates independently of the application, except through pre-arranged interactions. For example, a button widget knows how to draw itself, how to highlight itself when it is clicked on with the mouse, and how to respond to that mouse click.

The general behaviour of a widget, such as a **Push Button**, is defined as part of the Motif library. Xt defines certain base classes of widget, whose behaviour can be inherited and augmented or modified by other widget classes (subclasses). The base widget classes provide a common foundation for all Xt-based widget sets. A widget set, such as Motif's Xm library, defines a complete set of widget classes, sufficient for most user-interface needs. Xt also supports mechanisms for creating new widgets or modifying existing ones.

Xt also supports lighter-weight objects called gadgets, which for the most part look and act just like widgets, but their behaviour is actually provided by the manager widget that contains them. For example, a pulldown menu pane can be made up of button gadgets rather than button widgets, with the menu pane doing much of the work that would normally be done by the button widget.

Motif Push Button class inherits the ability to display a label from the Label widget class. The object-oriented approach of Xt completely insulates the application programmer from the code inside of widgets. As programmers, we only have access to functions that create, manage, and destroy widgets plus certain public widget variables known as *resources*. As a result, the internal implementation of a widget can change without requiring changes to the API.

Creating a widget is referred to as instantiating it. We ask the toolkit for an instance of a particular widget class, which can be customised by setting its resources. All Motif Push Button widgets have the ability to display a label; an instance of the Push Button widget class actually has a label that can be set with a resource.

Widgets are designed so that many of their resources can be modified by the user at run-time. When an application is run, Xt automatically loads data from a number of systems and user application files. The data from these files is used to build the resource database, which is used to configure the widget, in the application. If we want to keep the user from modifying resources, we can set their values when we create the widget. This practice is commonly referred to as hard-coding resources.

It is considered good practice to hard-code only those resource values that are essential to program operation and leave the rest of resources configurable. Default values for configurable resources are typically specified in an

application defaults file which is more colloquially referred to as the app-defaults file.

Motif widgets are prolific in their use of resources. For each widget class, there are many resources that neither the application nor the user should ever need to change. Some of these resources provide fine control over the three-dimensional appearance of Motif widgets.

The callback resources for a widget are a particularly important class of resources that must be set in the application code. A widget that expects to interact with an application provides a callback resources for each type of interaction it supports. An application associates a function with the callback resources in which it is interested; the function is invoked when the user performs certain actions in the widget. For example, a push button provides a callback for when the user activates the button. We must note however, that not every event that occurs in a widget results in a callback to an application function. Widgets are designed to handle many events themselves, with no interaction from the application. All widgets know how to draw themselves, for example. A widget may even provide application-like functionality. For example, a text widget typically provides a complete set of editing commands via internal widget functions called actions. Actions are mapped to events in a translation table. This table can be augmented, selectively overridden, or completely replaced by settings contained in the implementation of a widget class, in application code, or in a user's resources files.

In the basic Xt design, translations are intended to be configurable by the user. However, the purpose of Xt is to provide mechanism, not impose user interface policy. In Motif, translations are typically not modified by either the user or the application programmer. While it is possible for an application to

install event handlers or new translations and actions for a widget, most Motif widgets expect application interactions to occur only through callbacks. Some of the Motif widgets, particularly buttons when they are used in menus, have undefined behaviour when their translations are augmented or overridden. Motif widgets provides callback resources to support their expected behaviour. If a widget does not have a callback associated with an event to which we want our application to respond, we should be cautious about adding actions to the widget or modifying its translations.

B.7. The X-Designer

X-Designer is a graphical tool for building graphical user interfaces. It is not part of X structure but because of the way it works it can be considered to sit above the widgets set. X-Designer works with the standard OSF Motif toolkit, it is not a proprietary emulation. With X-Designer the user can use the full power and flexibility of Motif. X-Designer generates standard, portable C code and / or X resources files directly from the user's design. Code generated by X-Designer can be used on any platform which supports Motif. **Fig 1.1** displays the X-Designer screen when the tool is started up. Using X-Designer is easy. There are no new languages to learn, and the user interface is designed to get the maximum result from the fewest actions. X-Designer ensures that the widget hierarchies which the user creates are valid - the user cannot set an inappropriate child, and she cannot set an incorrect resource. With X-Designer, the user works on the widget hierarchy, and the tool shows what the result looks like. The structure of the design is always visible to the user, and one can cut, paste and move parts of the hierarchy to get the required result. Because the user works on the widget hierarchy, the design result (the dialog being designing) always behaves as the eventual interface will. The

user can try the effect of pressing buttons and so on, without having to switch into a special 'test' mode.

B.8. Futures of the X-Designer

- Good user Interface Techniques
- Complexity hidden
- (X Window System Manual 1000 pages)

  (X Toolkit Intrinsics Manual 600+ pages)

  (Motif Reference Manual 1000+ pages)
- Quick to prototype
- Clean code. (X-Designer generates standard, portable C code)
- Extensible
- Experimentation encouraged

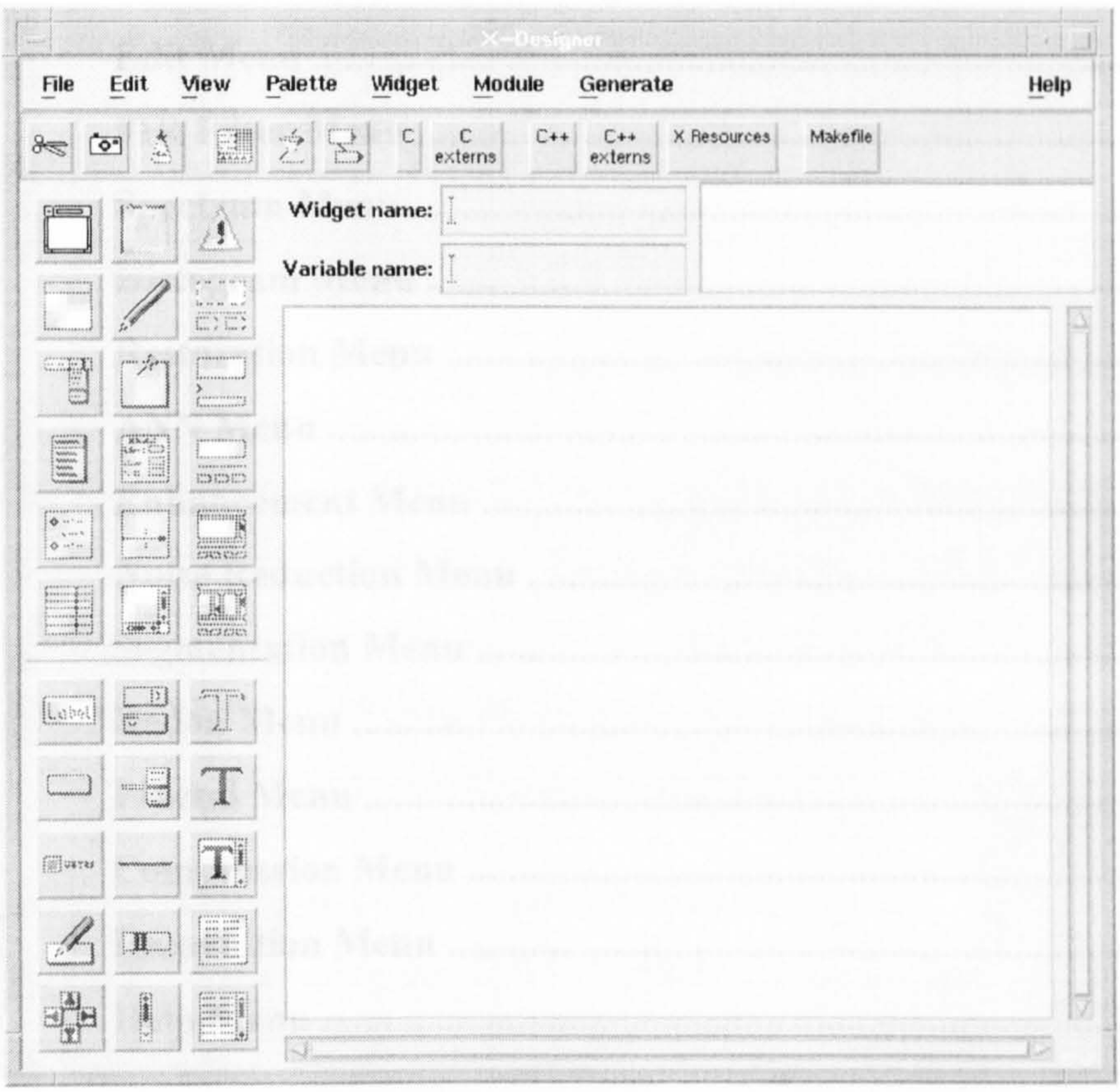


Fig 1.1 The X-Designer Main Window

Appendix C

User Manual to Software Development for The Thesis

This appendix explains how to start the application, states the interface features and describes the interface in detail.

In this chapter **Page No**

C.1. Running DIP

In order to start the application the user has to type DIP and press the Enter key in the UNIX prompt. The application's interface window pops up. **(Fig 5.1)**

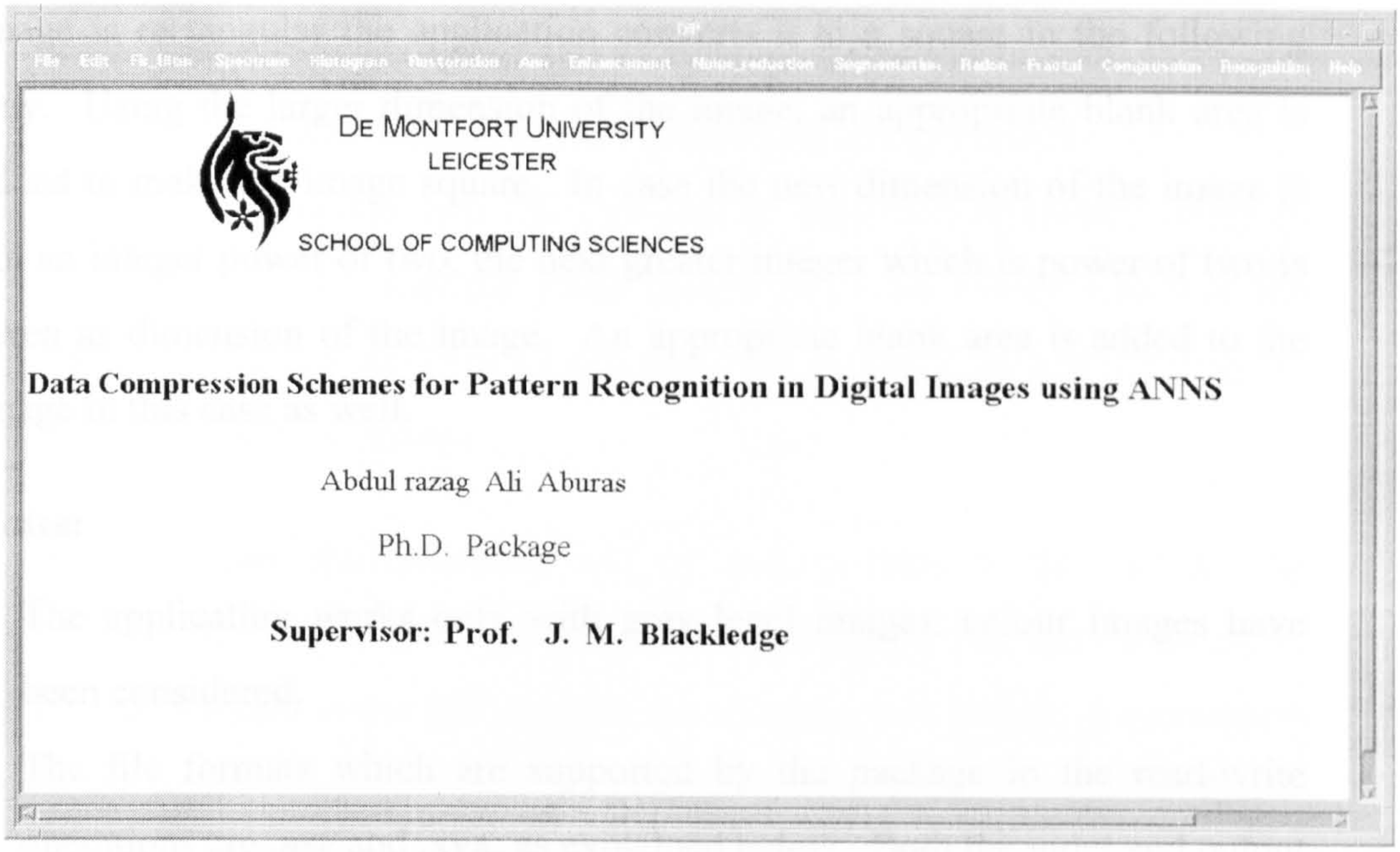


Fig 5.1 DIP's Application and Main Interface Window

The main window is composed of the menu bar, a horizontal and vertical scroll bar and the workspace of the main window. Until a file is loaded the File menu is the only one available and the label of **(Fig 5.1)** is displayed on the workspace. On pressing the File button the menu shown in **(Fig 5.2)** pops down with only the Open and Exit options available.

Using the open option the user can load an image on the workspace while using the Exit terminates the application. As soon as an image is loaded all the options in the menu bar become available.

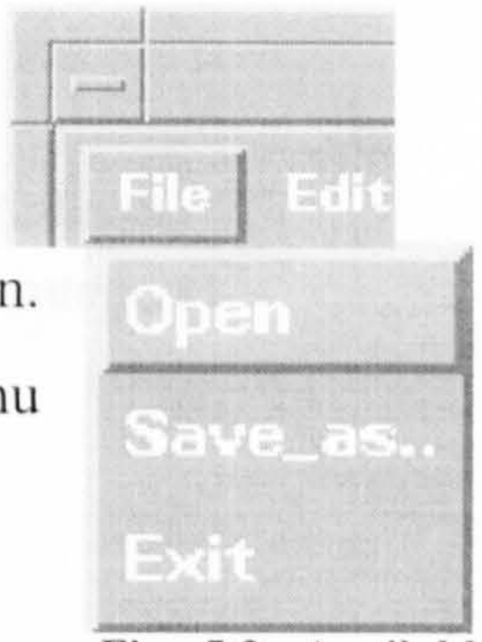


Fig 5.2 Available options in File Menu

C.2. Features of the DIP Package

Since the Fast Fourier Transform is used in most processes applied to an image, the application requires a square image whose dimension is a power of two. This feature **does not restrict** the user to load such images. If the loaded image is rectangular the application converts it to a square in the following way. Using the larger dimension of the image, an appropriate blank area is added to make the image square. In case the new dimension of the image is not an integer power of two, the next greater integer which is power of two is taken as dimension of the image. An appropriate blank area is added to the image in this case as well.

Notes:

- The application works only with grey level images, colour images have been considered.
- The file formats which are supported by the package in the read-write operations are **.asc** and **.xva**, as explained below. Both the input and output is in ascii form and thus readable by the user.

.asc format
Width of image (integer)
Height of image (integer)
Data representation (floats)

.xva format
P2 (format type descriptor)
Width of image (integer)
Height of image (integer)
Shades of grey in the image (integer)
Data representation (positive integers)

The first format has been implemented in order to read files which have been given as samples during the course of this research.

The second format has been developed in order to provide compatibility between the DIP and the xv application; xv is a program which provides image facilities and can be found in most of the systems using X windows. This kind of compatibility is very important in the current version of DIP because it is a way of loading images with different formats into our application. Until other formats are incorporated into DIP, xv can be used as a translator using its option to save a file with the PBM (ascii) greyscale format which is the equivalent of xva format. (The xv program runs by typing xv and pressing the Enter button in UNIX prompt).

A file must have the extension .asc or .xva in order to be read by the application. Any other file is not read. When saving a file, the extension of the format is added to the file name so the user may not include the extension in the file name. The user must not attempt to save a file if insufficient disk space is available, in this case the program will terminate with errors.

The picture which is in the workspace of the main window is the subject of the current process. The result of a specific process asked by the user is displayed on a window which pops up on the screen. **(Fig 5.3)**

The application opens up to six such sub-windows for different tasks. In a subsequent task, the picture which is displayed on the first sub-window pops up with changes. In this way the application keeps on changing the pictures when different processes take place.

The sub-windows have a menu bar with the File and Edit options.

In the Edit menu the user can find the Copy option, which copies the image which is displayed on the sub-window to the main window. In this way images which are displayed in the sub-windows can be used for further processing.

Fig 5.3 A sub-window

The File menu provides the Save as and Close options. The first allows the user to save the image which is displayed in the sub-windows; while the second allows her to close the sub-window to be closed. This and any other sub-window which have been closed will pop up again as required, so as to avoid changing the image in any visible sub-window.

C.3. Learning to Use the DIP Package

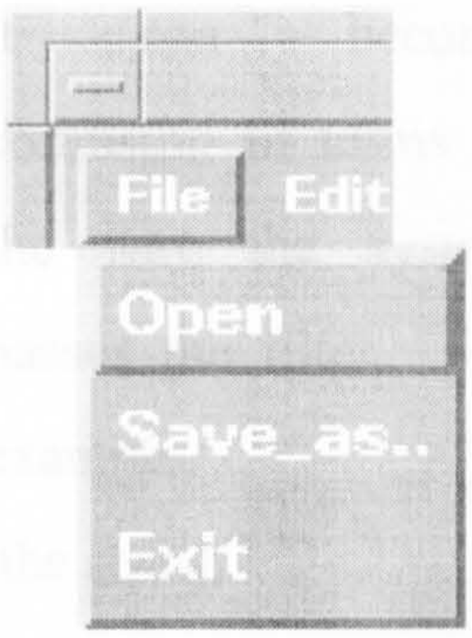


Fig 5.4 File Option

5.3.1. File Option Menu

Appears when the user clicks the **File** button on the menu bar or presses F10 **(Fig 5.4)**.

Open

Clicking the Open button (ctrl+0) brings up the File Selection box shown in (**Fig 5.5**). This box has four components. The filter text entry area specifies the directory and the filter. The directories list, displays the sub-directories of the current directory specified by the filter. If the user selects a directory, the filter is modified to reflect the selection. The files list shows the file in the

current directory. The selection entry area specifies the file selected by the user. If the user selects a file from the file list, the full pathname is displayed in the selection text entry area.

The File selection box has three available buttons in its action area. (The Help is not available).

If the **Cancel** button is pressed, the File Selection box is popped up.

The **Filter** button acts on the directory and pattern specified in the filter text entry area. While this process seems straightforward, it can become confusing in terms of the way the system passes the filter. For example,

the string

/usr/motif/lib/Xm

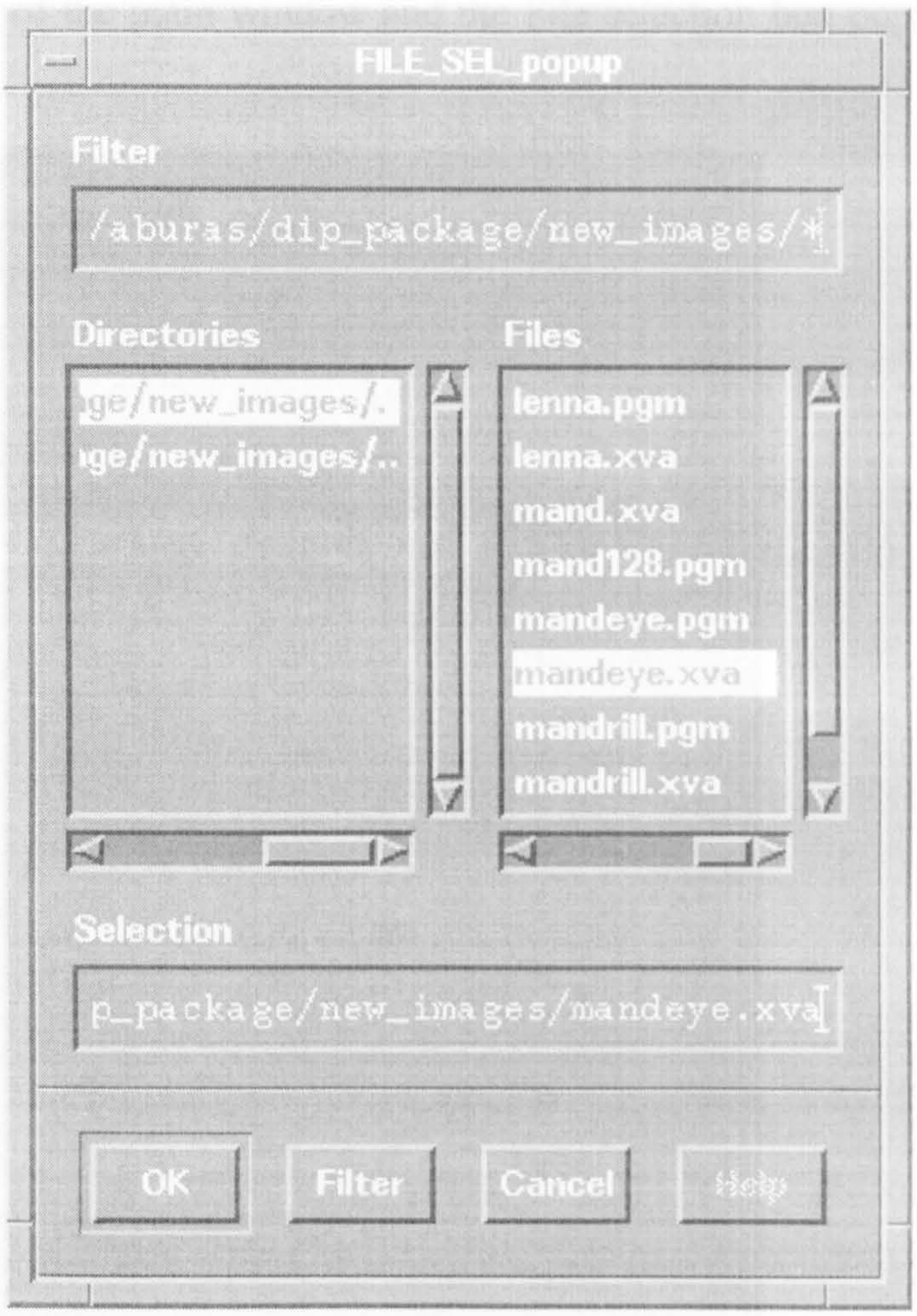


Fig 5.5 The File Selection Dialog Box

appears to be a common directory path, but in fact, is interpreted so that the directory is */usr/motif/lib* and the pattern is Xm. If searched, the directories list will contain all the directories in */usr/motif/lib* and the files list will not

contain anything because Xm is a directory, not a pattern that matches any files.

When the user selects the **OK** button the program loads the image file which has been selected. If the program does not support the format in which the file is written, an information dialog box is popped up. Otherwise the image is loaded in the workspace of the main window and the File selection box pops down.

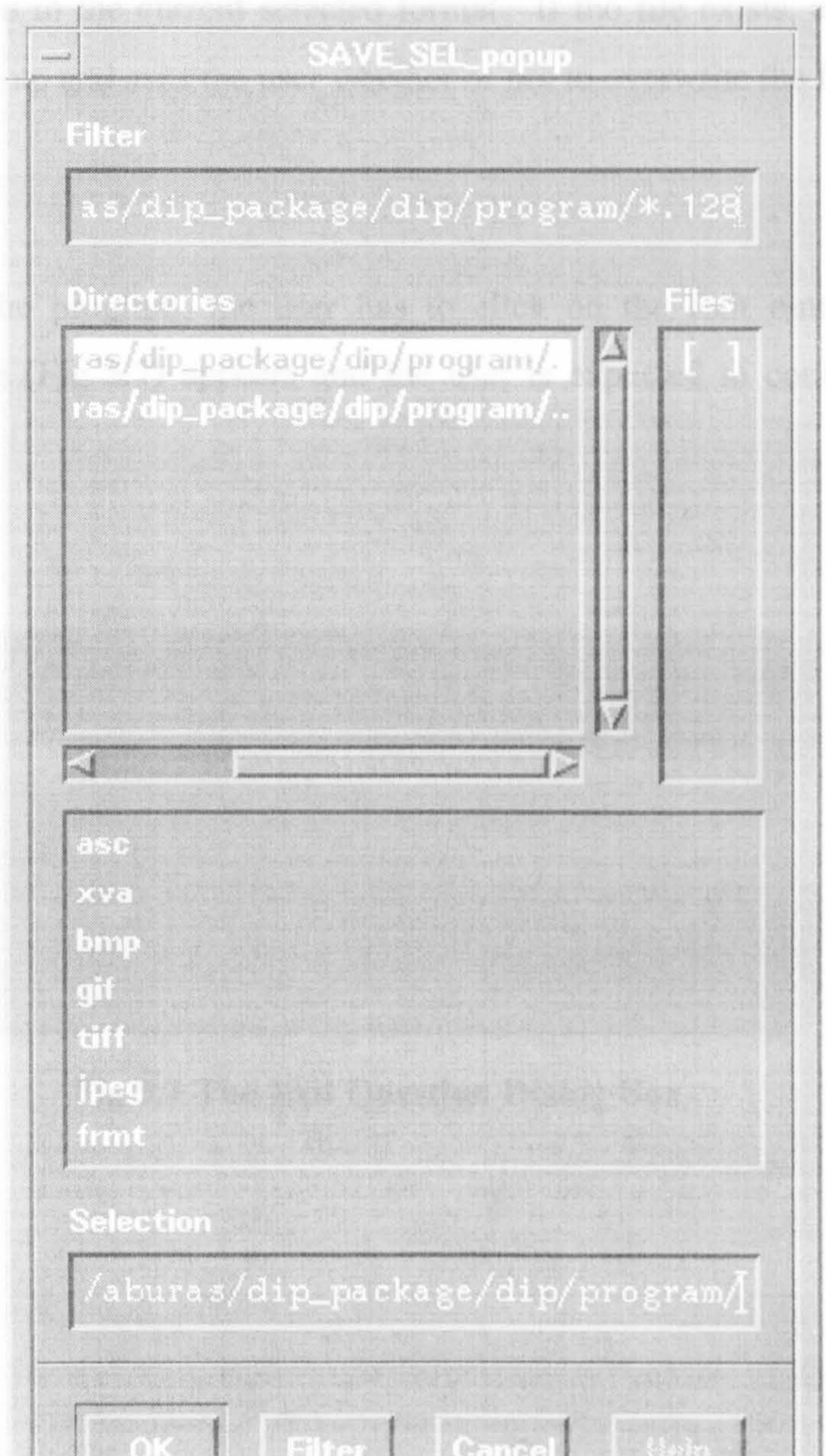


Fig 5.6 List of available formats in the File Selection Save Dialog Box

Save as

Clicking the Save as button brings up the file selection box. The new feature of this box, compared to the one described before in the File/Open options, is the list of available formats which includes (**Fig 5.6**). Selecting one of the formats, the filter is modified to reflect the selection. When the user presses the OK button, the image which is displayed in the workspace of the main window is saved in the current selected format. If the file exists, a question dialog box pops up and asks the user whether or not to overwrite the file.

Exit

To exit from the program, the user has to click on the Exit button. The question Dialog **(Fig 5.7)** appears and the user is expected to confirm their choice.

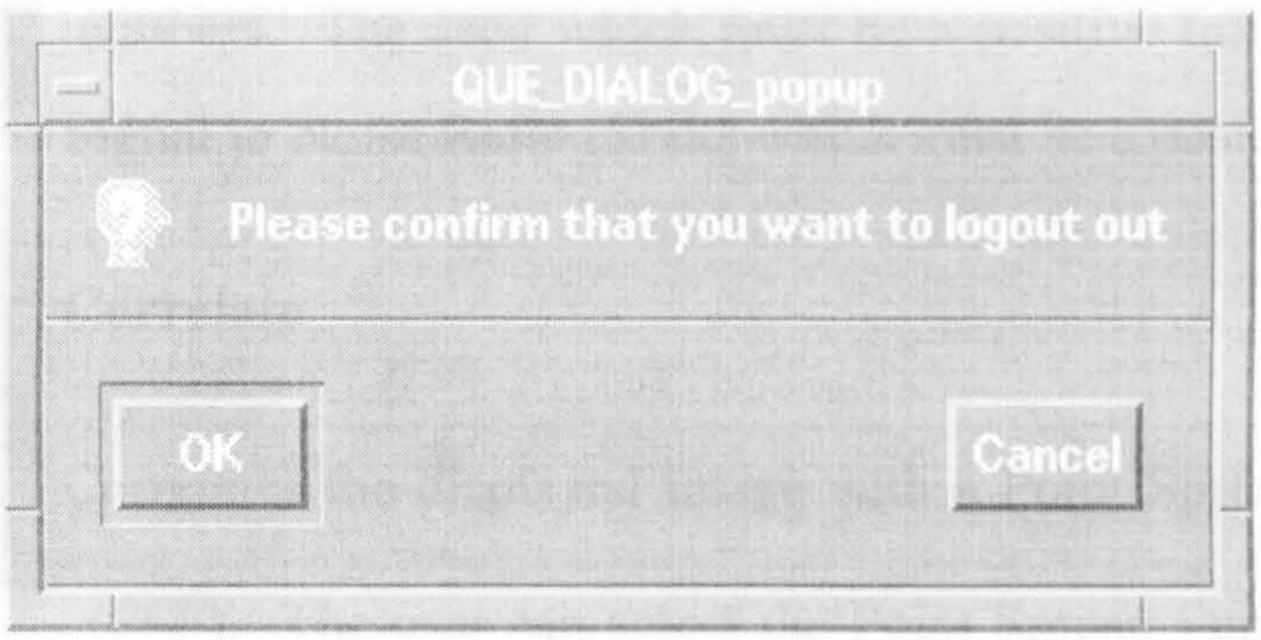


Fig 5.7 The Exit Question Dialog Box

C.3.2. Edit Option Menu

Appears when the user clicks the **Edit** button on the menu bar **(Fig 5.8).**

Copy

Copies the image which is displayed in the workspace of the main window to a sub-window.

Rotate

Rotates the image at an angle defined by the user. When the user presses the **Rotate** button, a box pops up where the angle of rotation in degrees (it is a float number) can be entered.

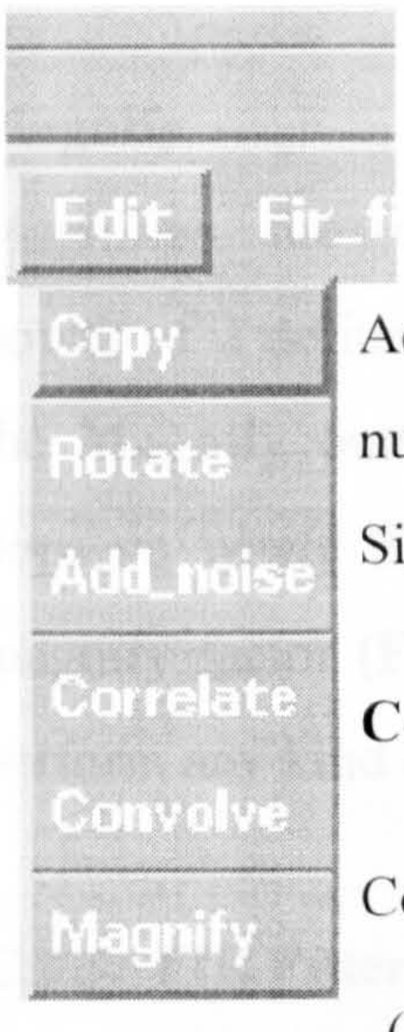


Fig 5.8
The Edit Menu

Add noise

Add noise to an image. The user is asked to type in two numbers. The seed which must be a positive integer and the Signal to Noise Ratio (SNR) which must be a positive float.

Correlate

Correlates the displayed image with a Point Spread Function (PSF). The user can select the Point Spread Function in the toggle box which pops up (**Fig 5.9)** when the **Correlate** button is pressed. Each choice raises a PSF box where the user is asked to enter the characteristics of the PSF. In the Gaussian box, the Gaussian width is required, it must be an integer greater than zero. In the elliptical disc box, the centre, the axes and the grey level of the disc are expected. All the values

must be integers. Twice the length of the largest disc axis must be less than or equal to the displayed image resolution. In the rectangular box, the user is asked to enter values for the centre, the sides and the grey level. All the values must be integers with the largest side of the rectangle less than or equal to the displayed image resolution.

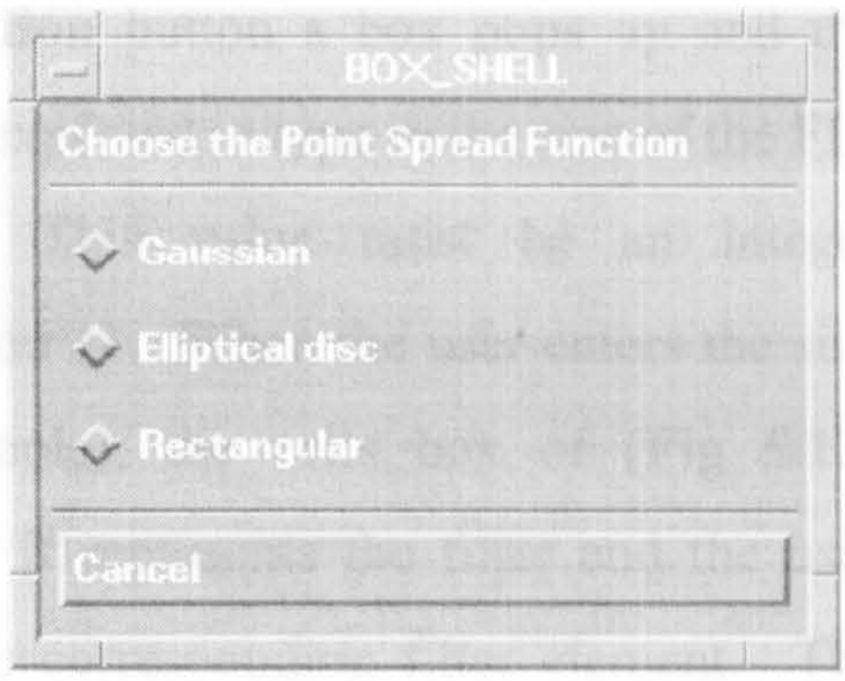


Fig 5.9 The PSF Toggle Box

Convolve

The same as is correlate above.

Magnify

Magnifies the displayed image by power of 2 defined by the user. When the **Magnify** button is pressed, a box pops up where the user can select the magnify factor **(Fig 5.10)**. It does not perform any kind of interpolation.

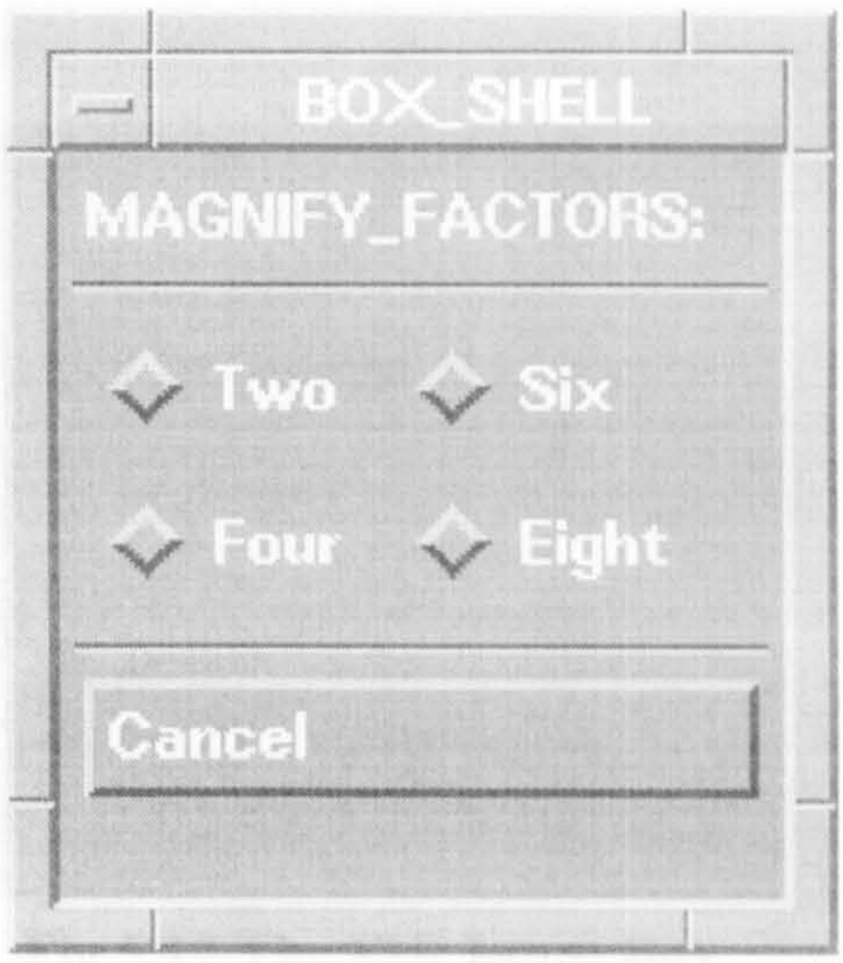


Fig 5.10 Magnify Power Box

C.3.3. FIR Filters Option Menu

Appears when the user clicks the FIR Filters button on the menu bar **(Fig 5.11).**

Correlation

Perform an FIR correlation of the image. When the user processes the

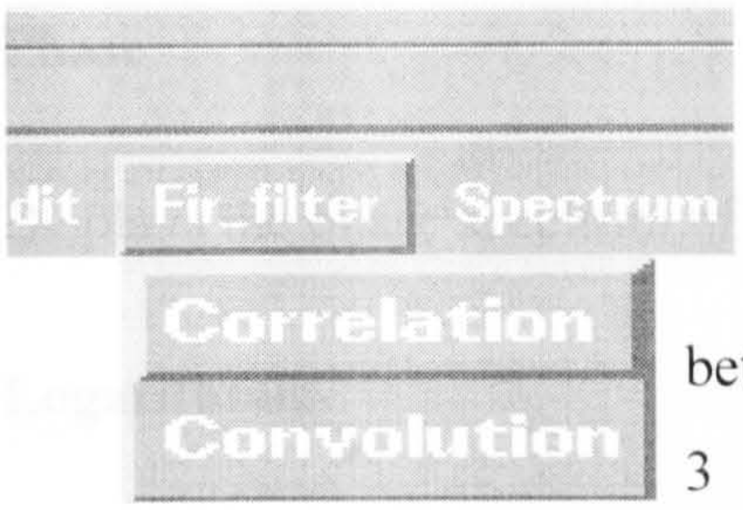


Fig 5.11 The FIR Filters Menu

correlation button a box pops up and the user is expected to type in the size of the FIR filter. This value must be an integer between 3 and 11. When the user enters the size 3 (for example), the cells box of (**Fig 5.12)** appears. It represents the filter and the user has to type in the cells the value of the corresponding filter element. The values may be floats. On pressing the OK button when all the values are correct, the FIR filtered image is computed; otherwise an information dialog box informs the user of a mistake.

Convolution

The same as correlate above.

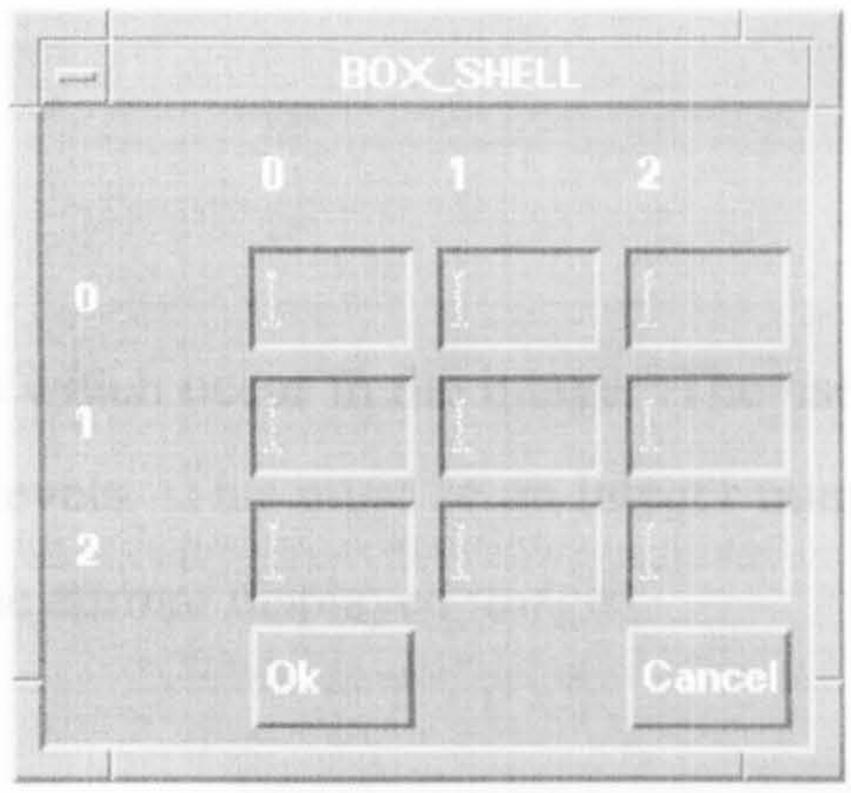


Fig 5.12 The FIR Input Box

C.3.4. Spectrum Option Menu

Appears when the user clicks the **Spectrum** button on the menu bar **(Fig 5.13).**

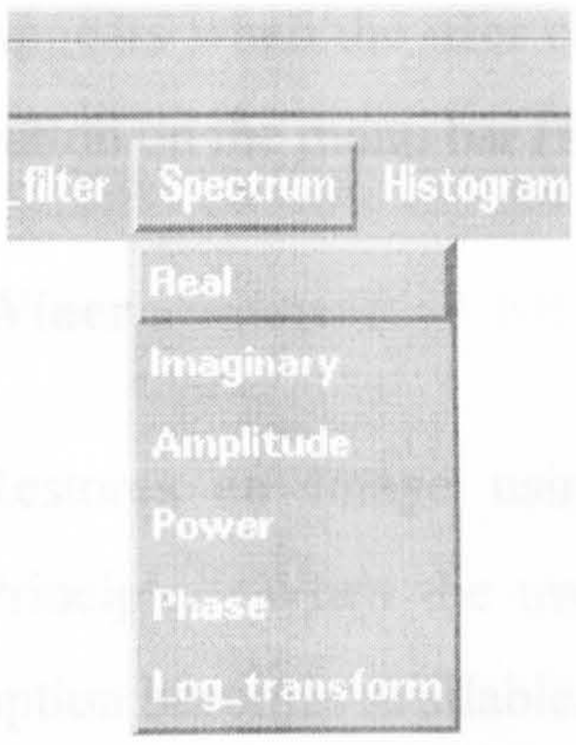


Fig 5.13 The Spectrum Menu

Real

Displays the real part of image spectrum.

Imaginary

Displays the imaginary part of image spectrum.

Amplitude

Displays the amplitude spectrum of the image.

Power Displays the power spectrum of the image.

Phase

Displays the phase spectrum of the image.

Logarithmic

Displays the logarithmically enhanced amplitude spectrum of the image.

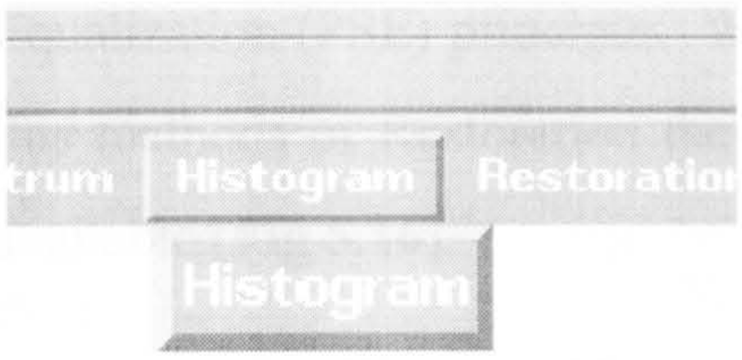


Fig 5.14 The Histogram Menu

C.3.5. The Histogram Option Menu

This pops down when the user clicks the **Histogram** button on the menu bar **(Fig 5.14)**.

Histogram

Displays a histogram of the grey levels which occur in the image. The user is prompted to enter the number of grey levels. This must be an integer number less than or equal to the resolution of the current displayed image.

C.3.6. Restoration Option Menu

Appears when the user clicks the **Restoration** button on the menu bar **(Fig 5.14)**.

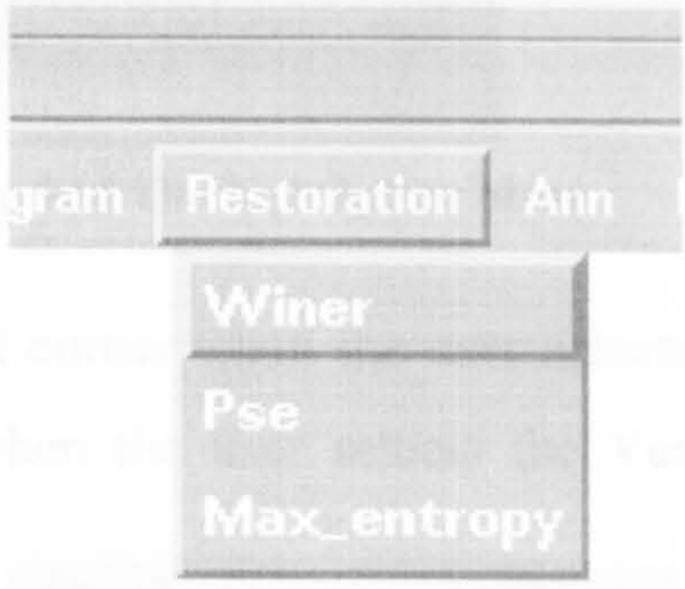


Fig 5.15 The Restoration Menu

Winer

Restores an image using the Least Square Principle. When the user selects this method of restoration, a Winer filter option becomes available. On selecting this option, the user is asked to choose a point spread function which is used for the restoration. The selection box is illustrated in **(Fig 5.9)**.

The PSF possible choices are those described in the **Edit Menu/Correlate Option**. In this case, the user is asked to provide one more parameter which is the estimation of the SNR and may be a float.

Pse

Restores an image based on the Power Spectrum Equalization (PSE) principle. When the user selects this methods of Restoration the PSE filter becomes available **(Fig 5.16)**.

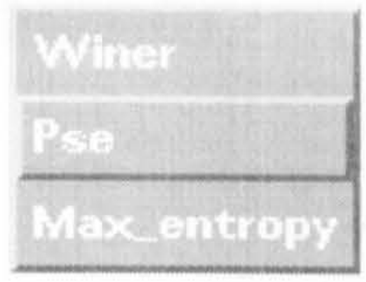


Fig 5.16 The Pse Option in the Restoration Menu

Max-entropy

Restores an image using the Maximum Entropy (ME) principle. A linearized ME filter is the available option when this kind of restoration is selected.

C.3.7. Ann Option Menu

Pops down when the user clicks the Ann button on the menu bar **(Fig 5.17)**.

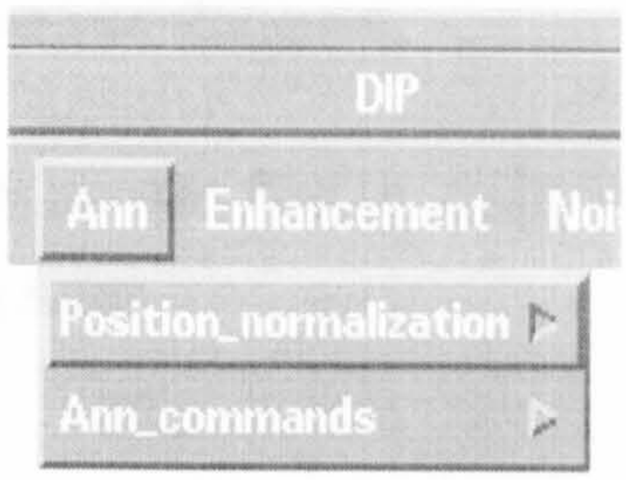


Fig 5.17 The Ann Option Menu

Position Normalisation

Used for plotting an image either at the top left corner when the user selects the No option or at any position (Random) when the user selects the Yes option. **(Fig 5.18).**

Fig 5.18 The Position Normalisation Option

Ann Commands

Used to deal with the Artificial Neural Networks commands.

The ANN commands are given in **(Fig 5.19)**.

Initial-ann

Initialises the Ann i.e. set all Ann to zeros.

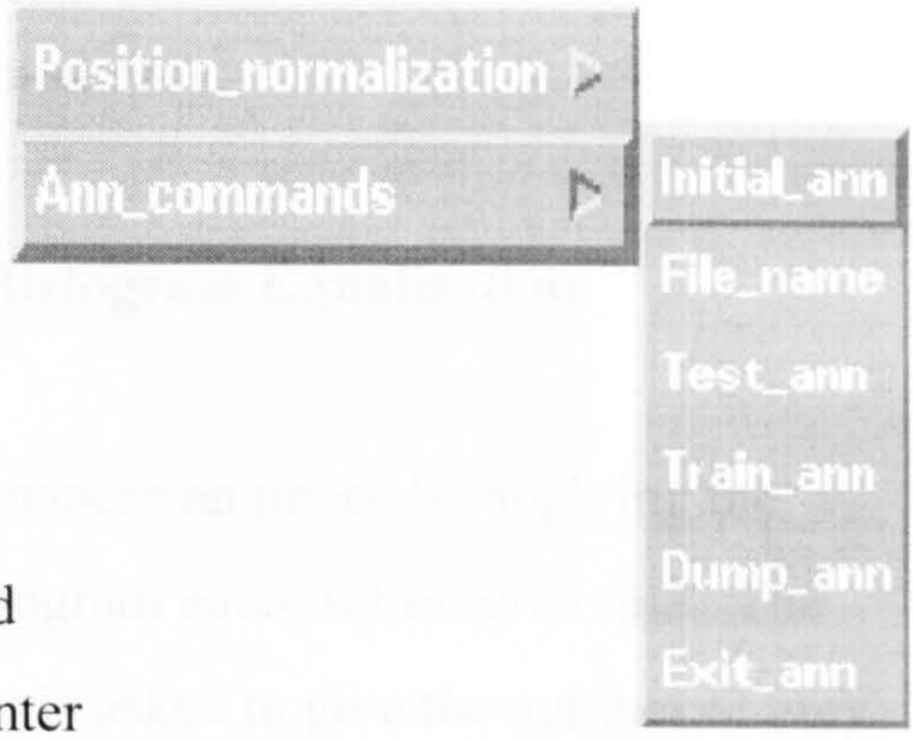


Fig 5.19 Ann Command Options

File-name

Opens and reads the file name formatted for the ANN. The user is prompted to enter the file name.

Test-ann

This is used to test the ANN. The user must give the number of patterns which must be a positive integer greater than or equal to 5000. This is needed to reach the target output. **(Fig 5.20)**

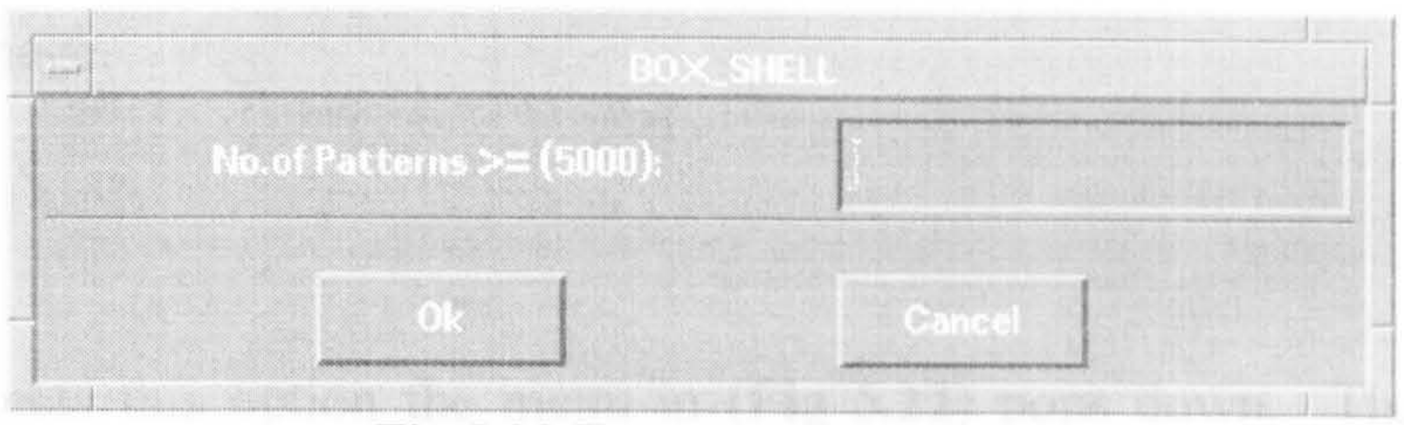


Fig 5.20 Test-ann dialog box

Train-ann

Same as above except that it is used to train ANN.

Dump-ann

Used to print out all the results from the ANN on the monitor screen.

Exit-ann

Used to terminate and exit from the ANN software program.

C.3.8. Enhancement Option Menu

The image enhancement menu **(Fig 5.21)** becomes available when the user clicks the **Enhancement** button on the menu bar.

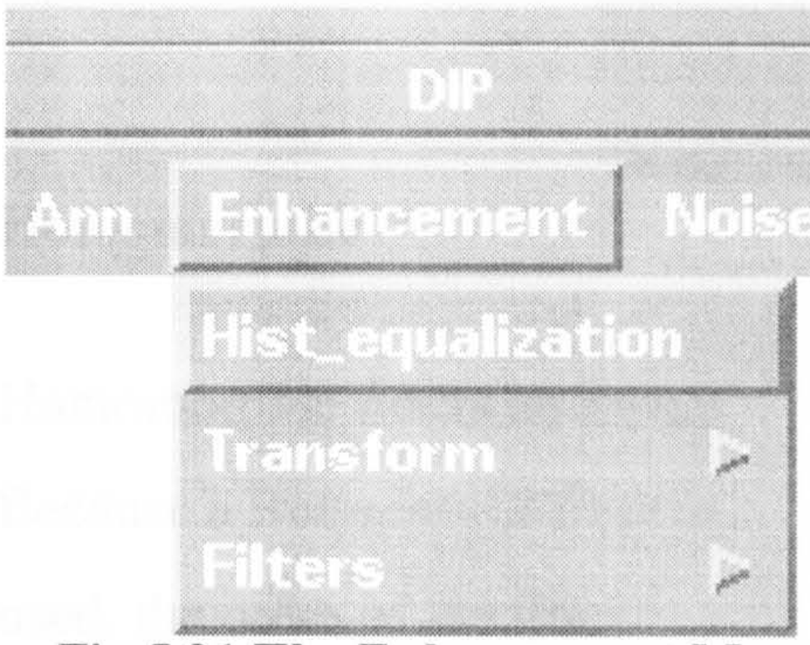


Fig 5.21 The Enhancement Menu

Histogram Equalisation

Enhances an image by applying the histogram equalisation principle. The user is asked to give the number of grey levels in the image which must be an integer less than or equal to the resolution of the image.

Transform

When the user selects this option the menu in **(Fig 5.22)** pops down. The logarithmic and the exponential transform options are available.

Logarithmic

The scaling parameter which is used in the transform must be defined by the user. The value must be a float greater than zero.

Exponential

As above.

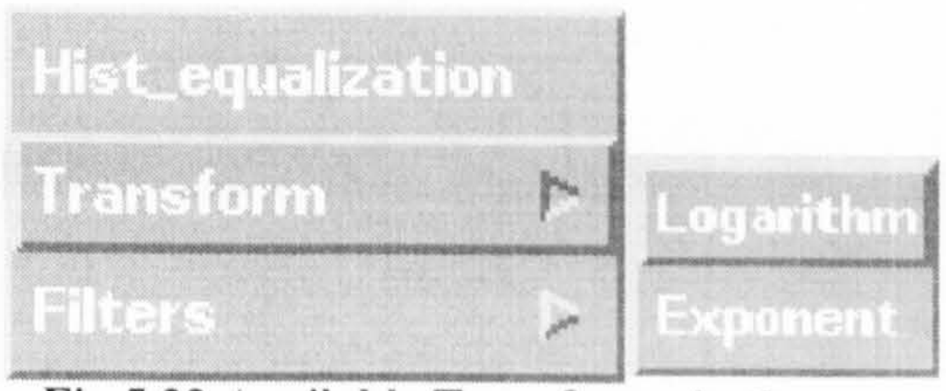


Fig 5.22 Available Transforms in the Enhancement Menu

Filtering

Enhances an image by applying different filtering techniques. The available

options are those illustrated in **(Fig 5.23)**.

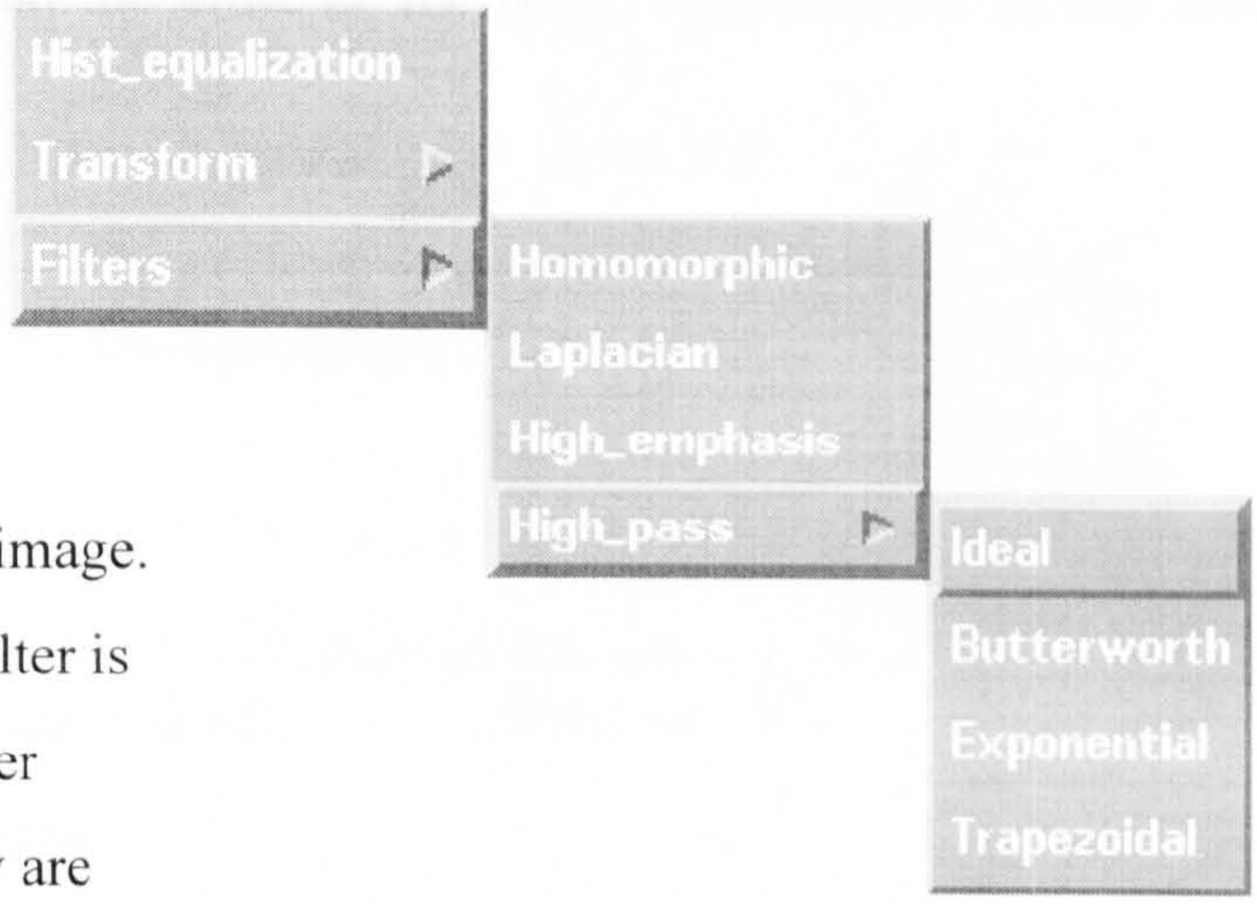


Fig 5.23 Filtering Options in the Enhancement Menu

Homomorphic

Homomorphic filters an image. Because a Butterworth filter is used, the order of the filter and the cut-off frequency are requested when the user selects this option. Both of these values must be integers greater than zero.

Laplacian

Laplacian filters an image.

High Emphasis

As with the Laplacian filter. The user must give the scaling factor which can be a float.

High pass

Highpass filters an image. Ideal, Butterworth, Exponential and Trapezoidal filter are available.

Ideal

The cut-off frequency, a positive integer number, must be given by the user.

Butterworth

The cut-off frequency and the order of the filter, both integers greater than zero, are requested.

Exponential

Same as above.

Trapezoidal

Same as Ideal Filter.

C.3.9. Noise Reduction Option Menu

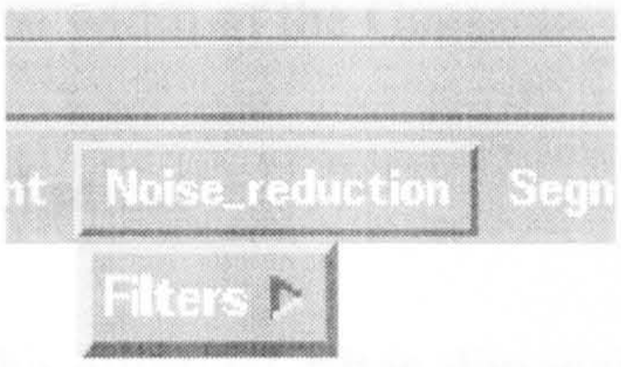


Fig 5.24 Noise Reduction Menu

This pops down when the user clicks the **Noise Reduction** button on the menu bar **(Fig 5.24)**.

Filtering

For noise reduction purposes, filtering techniques are revealed when the user presses the Filtering button **(Fig 5.25)**.

Median

Selecting this option the user is asked to give the size of the neighbourhood to be used. The value must be an odd integer.

Mean

As above, the user is asked for the neighbourhood size.

Lowpass

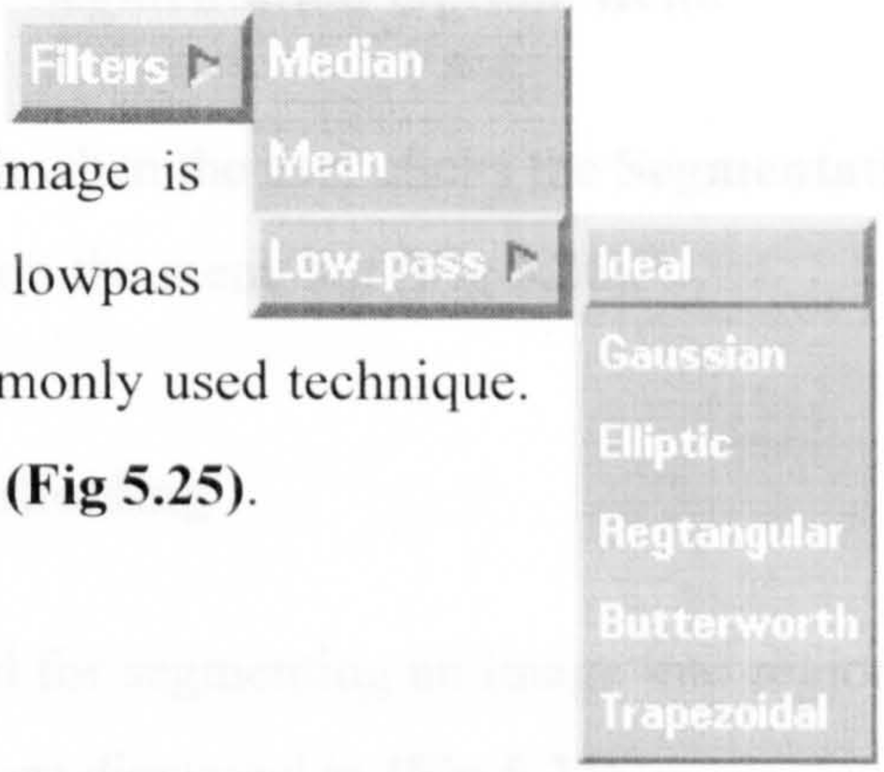


Fig 5.25 Filtering Options in the Noise Reduction Menu

Since in many cases noise in an image is primarily a high frequency effect, lowpass filtering for noise reduction is a commonly used technique. The available filters are illustrated in **(Fig 5.25)**.

Ideal

The user must give the cut-off frequency, a positive integer.

Gaussian

The width of the Gaussian, a positive integer, must be provided.

Elliptical

The elliptical filter dimensions in pixels are expected. Accepted values are integer numbers which are less than or equal to the image resolution.

Rectangular

As above.

Butterworth

See enhancement menu/filtering/homomorphic option.

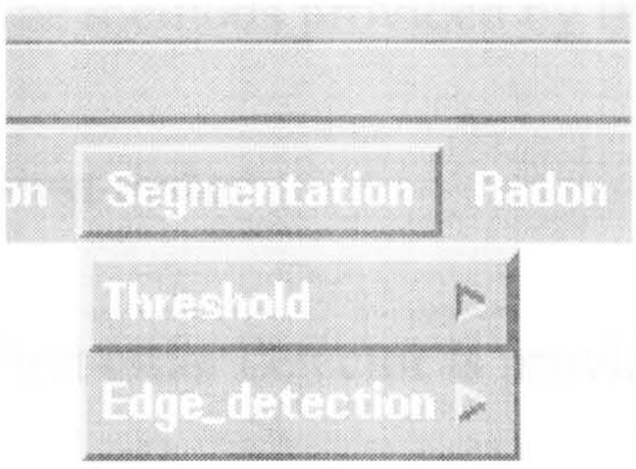


Fig 5.26 The Segmentation Menu

C.3.10. Segmentation Option Menu

Appears when the user clicks the **Segmentation** button on the menu bar **(Fig 5.26).**

Thresholding

Used for segmenting an image into regions of similarity. The available techniques are displayed in **(Fig 5.27).**

Single band

The user must provide the thresholding value, a float number.

Semi-threshold

The user must provide the semi-threshold value which must be a positive float less than 1.

Multi band

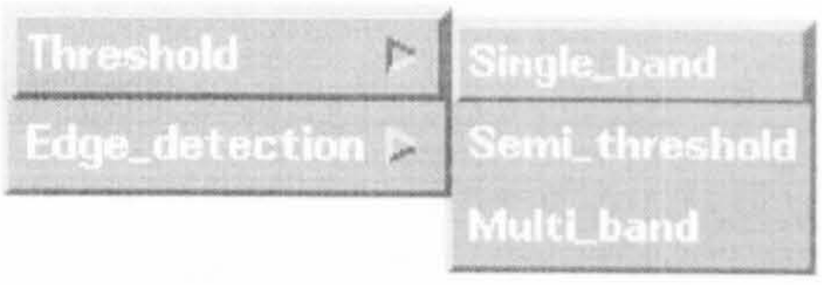


Fig 5.27 Thresholding Options

The user is prompted to select from a toggle button box the threshold's number, the maximum available number is six. As soon as the selection is made, an input data box pops up where the user is expected to type in the threshold values. The values must be positive floats.

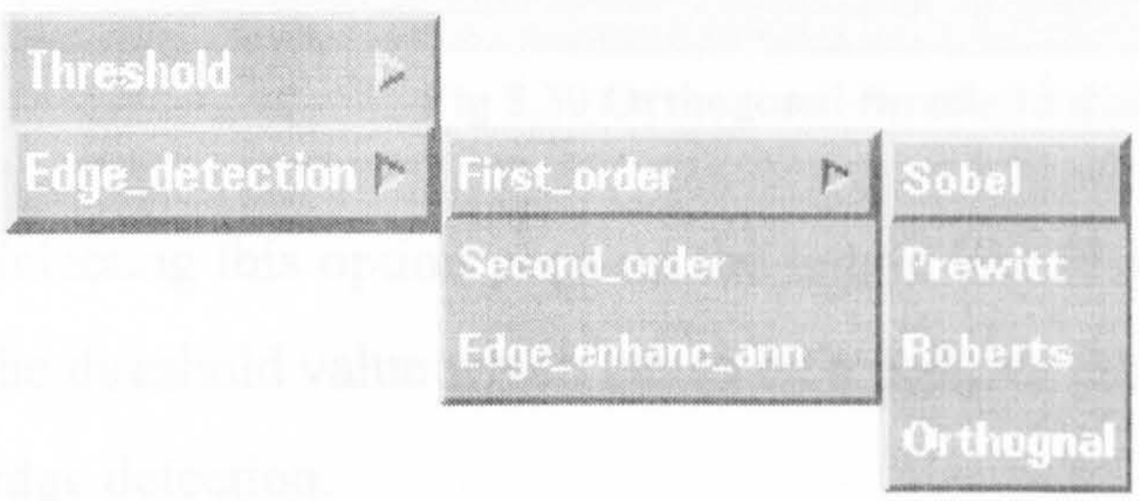


Fig 5.28 Edge detection Options

Edge detection

Segments an image into regions of discontinuity.

The methods provided by the program are illustrated in **(Fig 5.28)**.

First order

First order detector is provided for the edge detection.

Sobel

Use the sobel operator. Selecting this option pops up the toggle box **(Fig 5.29)** where the user has to select the direction of the edge detection.

Prewitt

Operates as the sobel operator except that the prewitt operator is used.

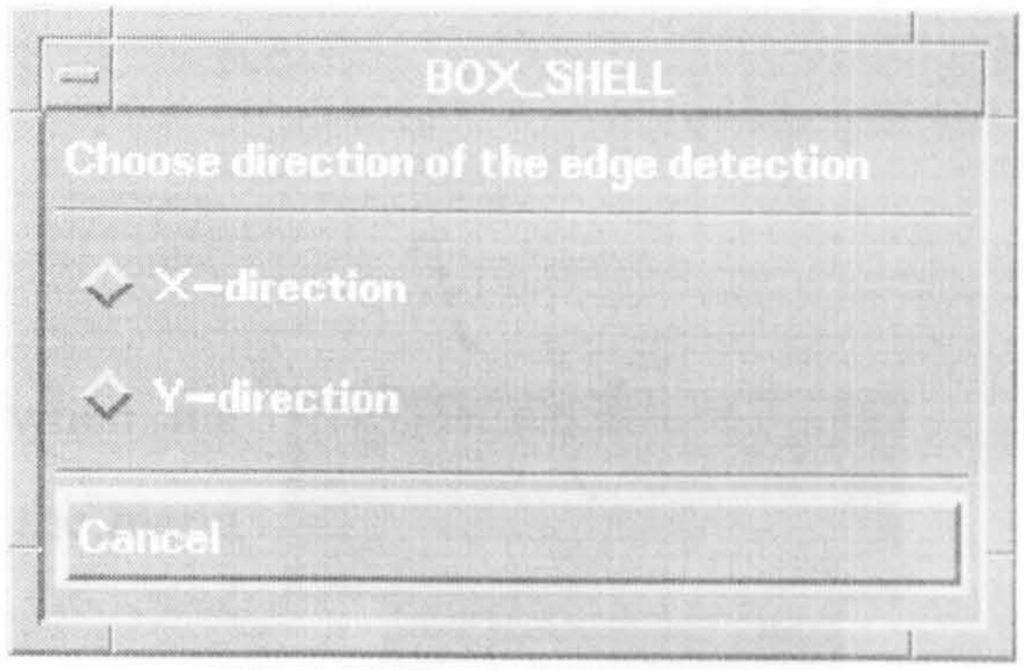


Fig 5.29 Edge detection box

Roberts

Roberts operator is provided for edge detection.

Orthogonal

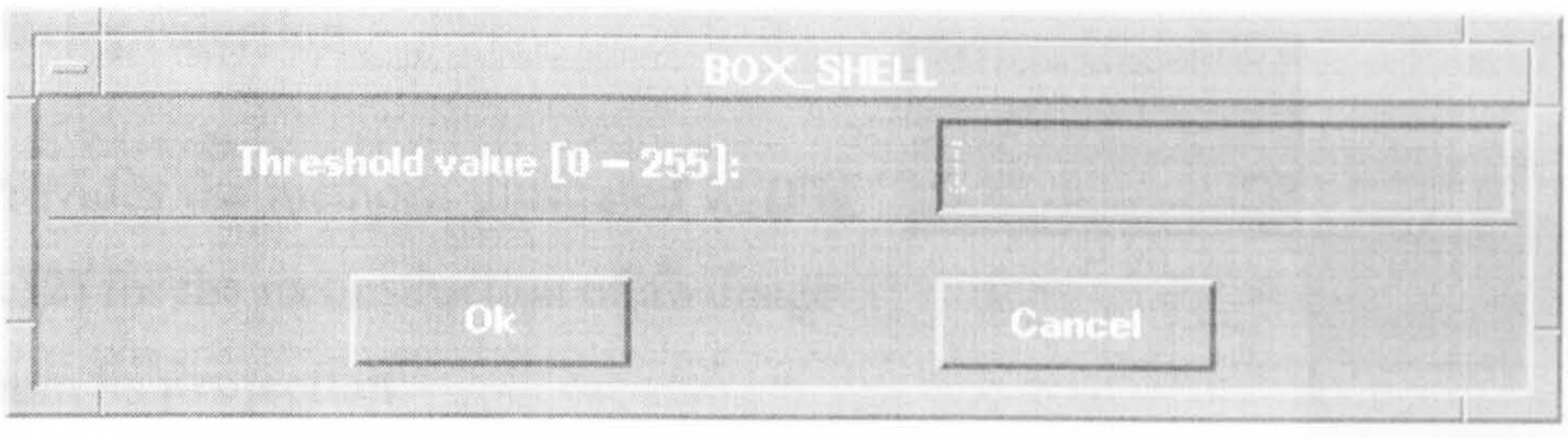


Fig 5.30 Orthogonal threshold dialog box

Selecting this option pops up the toggle box **(Fig 5.30)** where the user selects the threshold value which has to be a positive integer and must be provided for edge detection.

Atan

An edge detector using the tan inverse operator.

Second order

Second order detectors are provided for edge detection.

Edge Enhance-ANN

Edge Enhance-ANN is provided for edge detection using an Artificial Neural Network.

C.3.11. Radon Option Menu

The Radon menu pops down when the user clicks the Radon button on the menu bar **(Fig 5.31).**

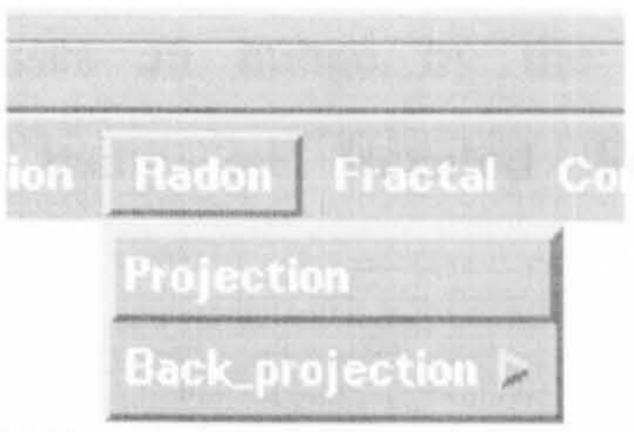


Fig 5.31 The Radon Menu

Projection

Computes the projection of an image.

Back-projection

Provides the methods illustrated in **(Fig 5.32)** for the reconstruction of an image from its **projection**.

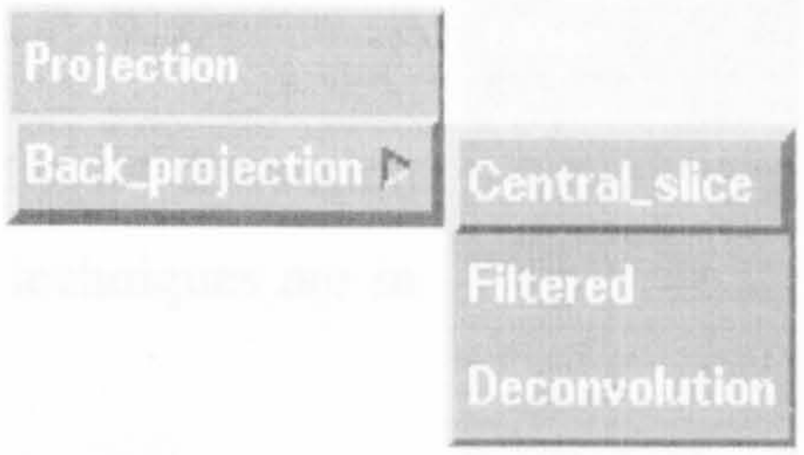


Fig 5.32 Methods for image reconstruction

Central Slice

Reconstructs an image using the Central Slice theorem.

Filtered

Reconstructs an image using filtered back projection method.

Deconvolution

Reconstructs an image from its projection by back-projecting and then deconvoluing.

C.3.12. Fractal Option Menu

This pops down when the user clicks the **Fractal** button on the menu bar **(Fig 5.33)**.

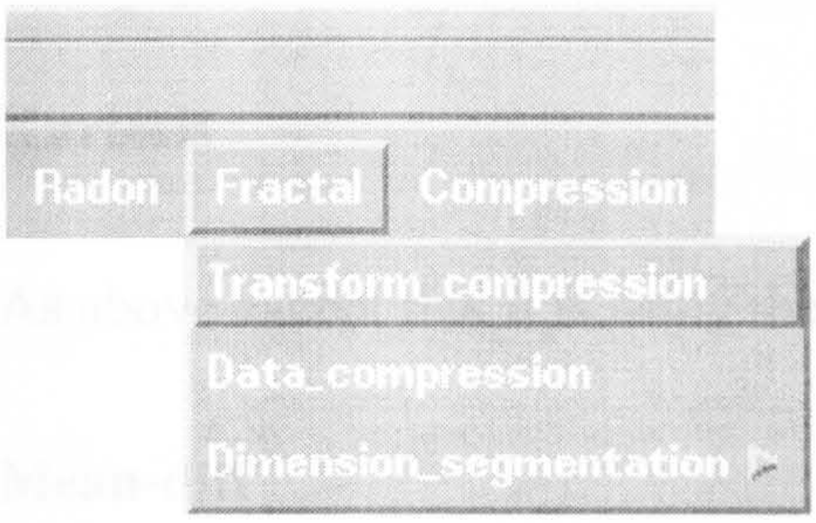


Fig 5.33 The Fractal Menu

Transform-Compression

Compresses an image by the **Fractal** method using an **Iterated Function System**.

Data Compression

Has not been provided.

Dimension-Segmentation

Used for segmenting and computing the fractal dimension (D) of an image (Texture). The available techniques are in **(Fig 5.34)**.

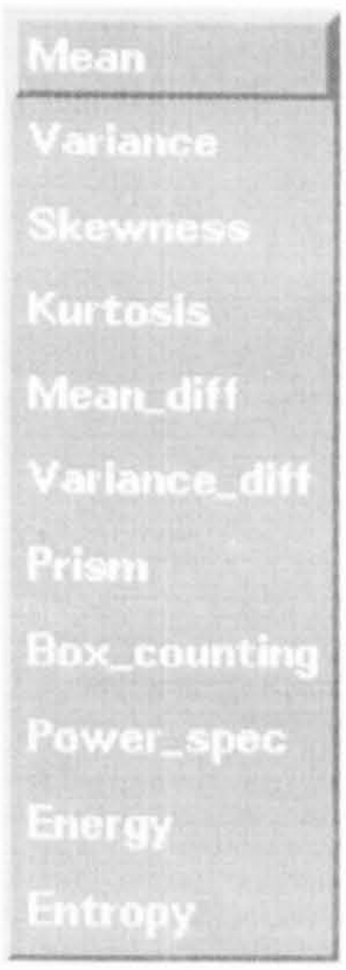


Fig 5.34 Fractal dimension-segmentation techniques

Mean

Computes the fractal dimension segmentation of an image using the **Mean** method. The user selects this method of

dimension-segmentation and is asked to choose the segmented window size as shown in **(Fig 5.35)**.

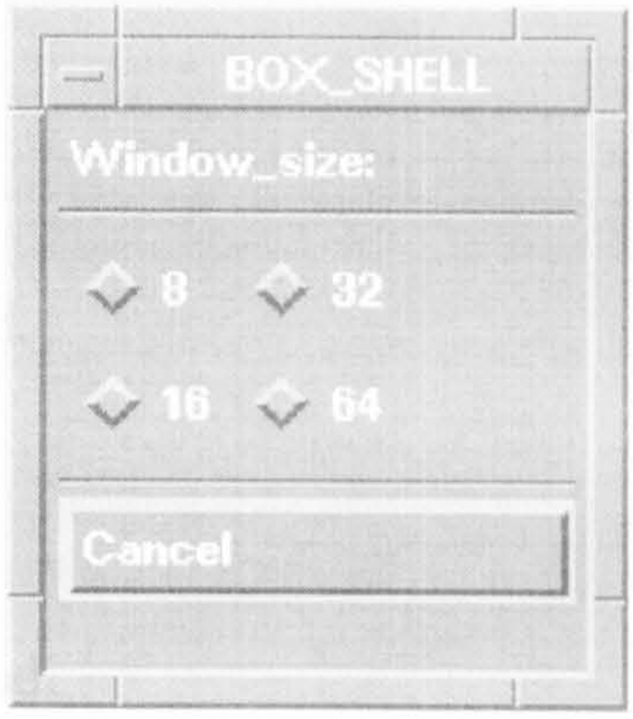


Fig 5.35 The Segmented window size

Variance

Computes the fractal dimension segmentation of an image using the **Variance** method. The segmented window size is shown in (**Fig 5.35)**.

Skewness

As above except that it is using the **Skewness** method.

Kurtosis

As above except that it is using the **Kurtosis** method.

Mean-diff

As above except that it is using the **Mean-diff** method.

Variance-diff

As above except that it is using the **Variance-diff** method.

Prism

As above except that it is using the **Prism** method.

Box-counting

As above except that it is using the **Box-counting** method.

Power-spec

As above except that it is using the **Power-spec** method.

Energy

As above except that it is using the **Energy** method.

Entropy

As above except that it is using the **Entropy** method.

Note: All the above methods are explained with detail in the Mathematical background section given in **Chapter 6**.

C.3.13. Compression Option Menu

The compression menu pops down when the user clicks the compression button on the menu bar **(Fig 5.36)**.

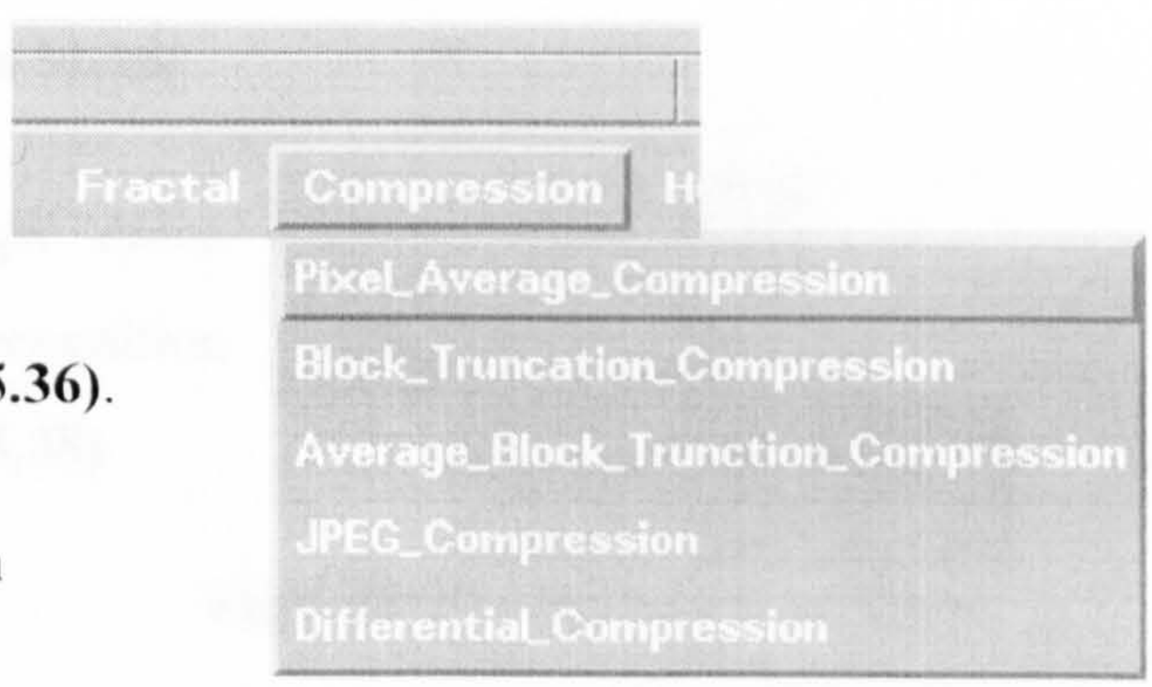


Fig 5.36 The Compression Menu

Pixel-Average-Compression

Provides the simplest method to compress an image. The user must choose the compression ratio from a dialog shown in **(Fig 5.37)** before compressing the image.

Block-Trunction-Compression

The Block-Trunction-Compression is used to compress an image.

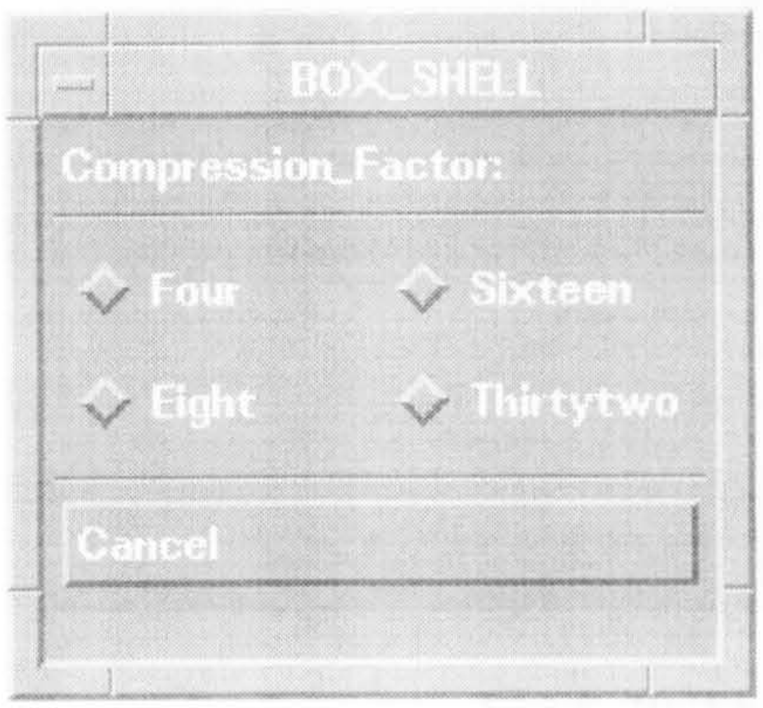


Fig 5.37 The Compression Ratio Dialog Box

Average-Block-Trunction-Compression

When the user selects this option, first the Block-Trunction-Compression is used followed by Pixel-Average-Compression.

JPEG-Compression

The JPEG-Compression is the most popular algorithm used to compress an image.

Differential-Compression

The Differential-Compression technique is used to compress an image when the user clicks this option.

C.3.14. Recognition Option Menu

The recognition menu pops down when the user clicks the recognition button on the menu bar **(Fig 5.38)**.

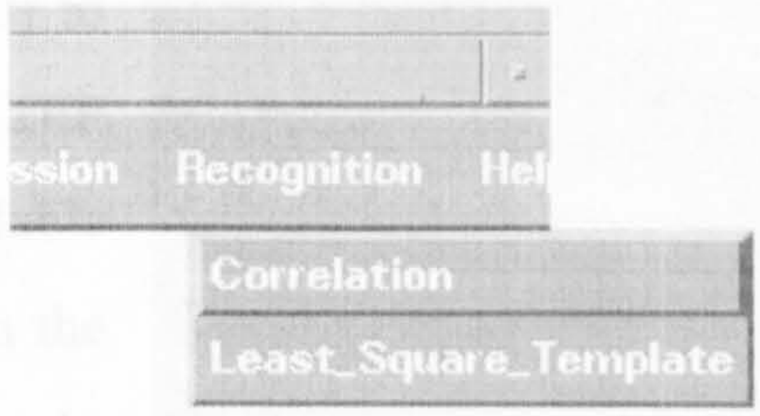


Fig 5.38 The Recognition Menu

Correlation

A standard method used to recognise an object in the image is by computing the correlation between the pattern and the object. The user must choose the pattern window size and the x, y co-ordinates (top left corner) of the pattern. **(Fig 5.39)**

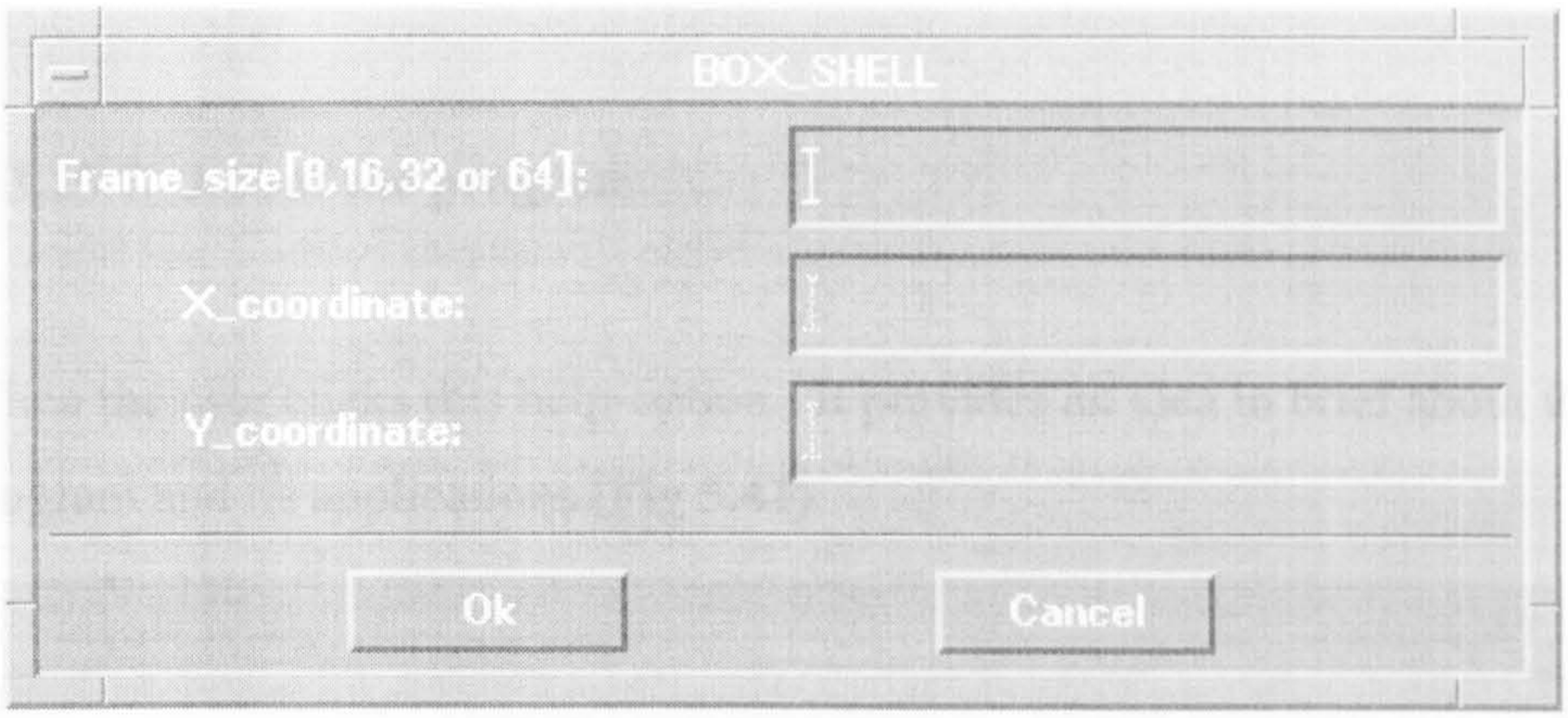


Fig 5.39 The Recognition Dialog Box

Least-square template

This provides the simplest method to recognise an object in the image by using least-square analysis template method. The user must choose the pattern window size and x, y co-ordinates (top left corner) of the pattern. **(Fig 5.39)**

C.3.15. Help Option Menu

This is the on-line-help menu **(Fig 5.40)** to assist the user when he/she is using the DIP package. The help menu becomes available when the user clicks the **Help** button on the menu bar. All the help menu and the sub-menu help options are completely independent from the main menu of the DIP package. In other words, the user can display any help option available in **parallel** with the other options in the main menu.

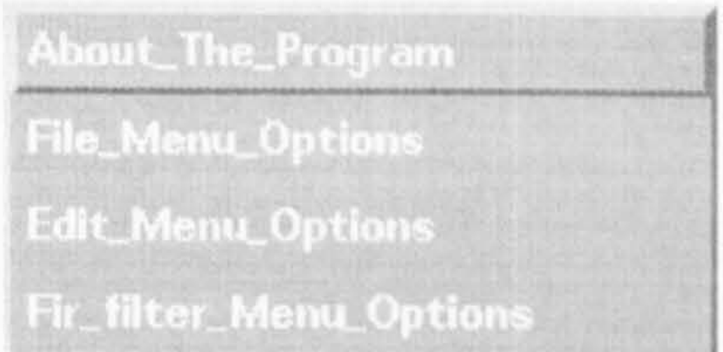


Fig 5.40 The On-line-help menu Options

C.3.15.1. About the program

When the user clicks this help-option. It provides an idea in brief about the program and its applications **(Fig 5.41)**.

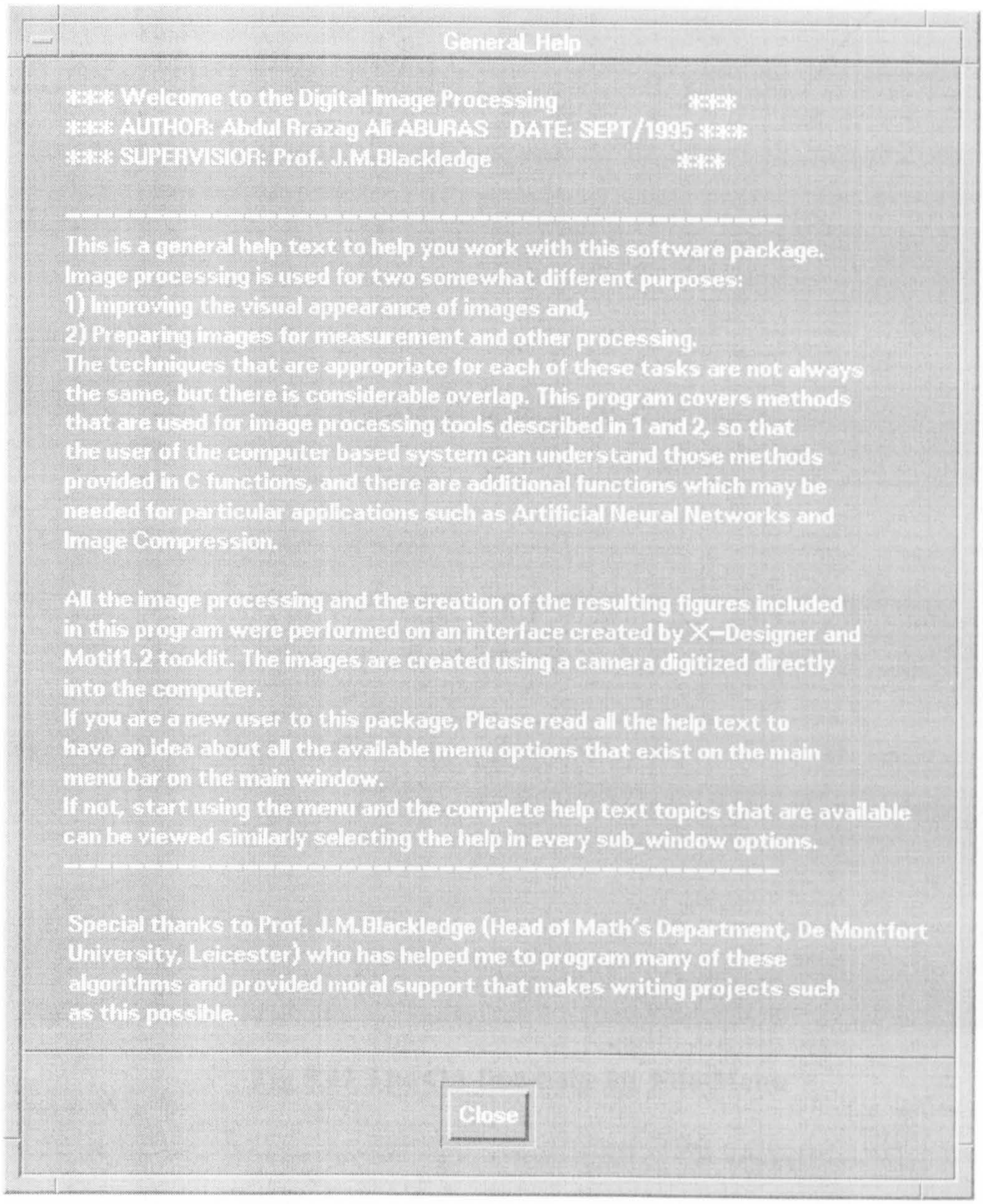


Fig 5.41 The On-line-help option (About the program)

C.3.15.2. File-Menu Commands

Provides on-line-help for the File-menu when the user selects this option **(Fig 5.42)**.

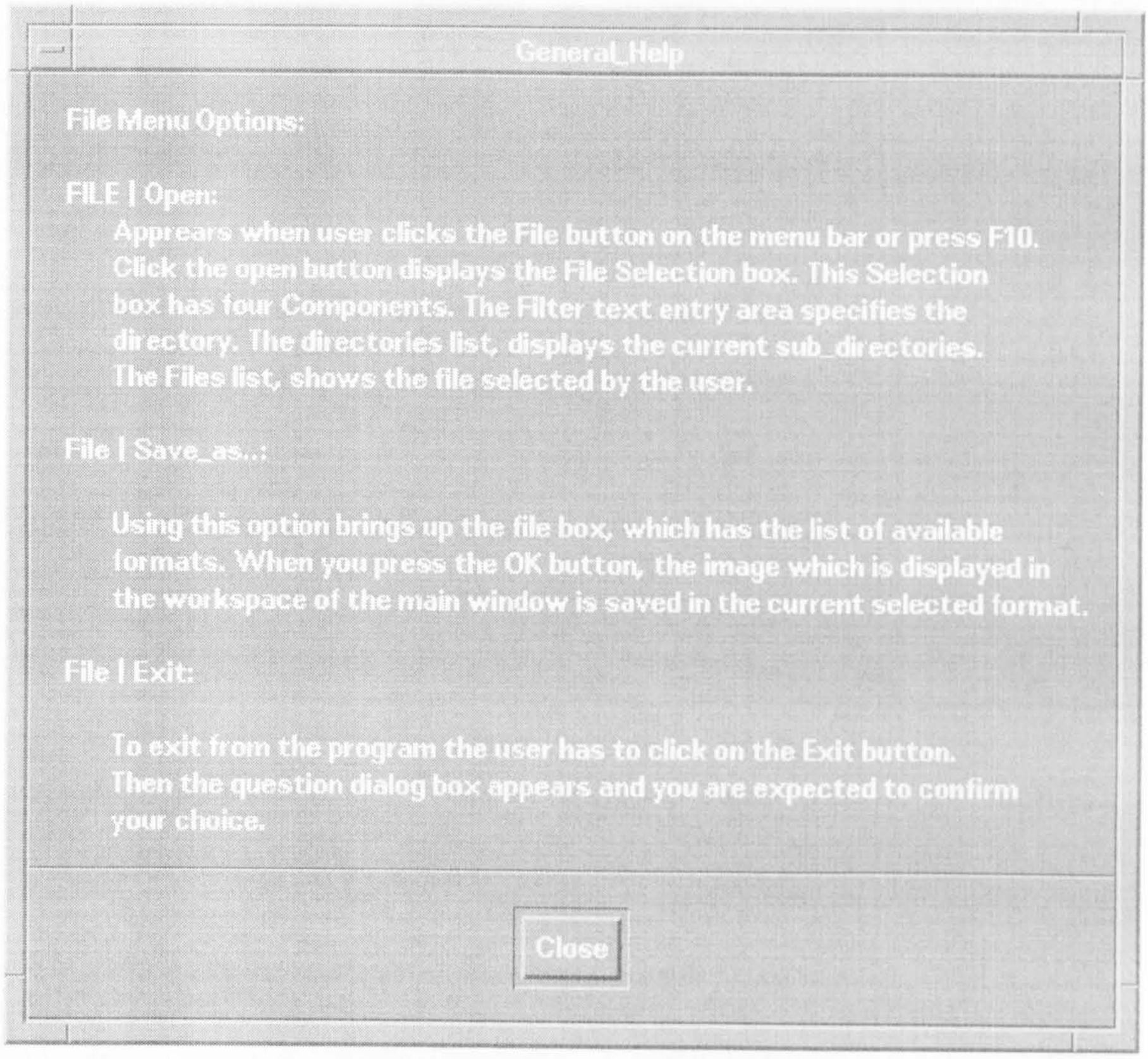


Fig 5.41 The On-line-help for File-Menu

C.3.15.3. Edit-Menu Commands

Provides on-line-help for the edit menu when the user selects this option **(Fig 5.43)**.

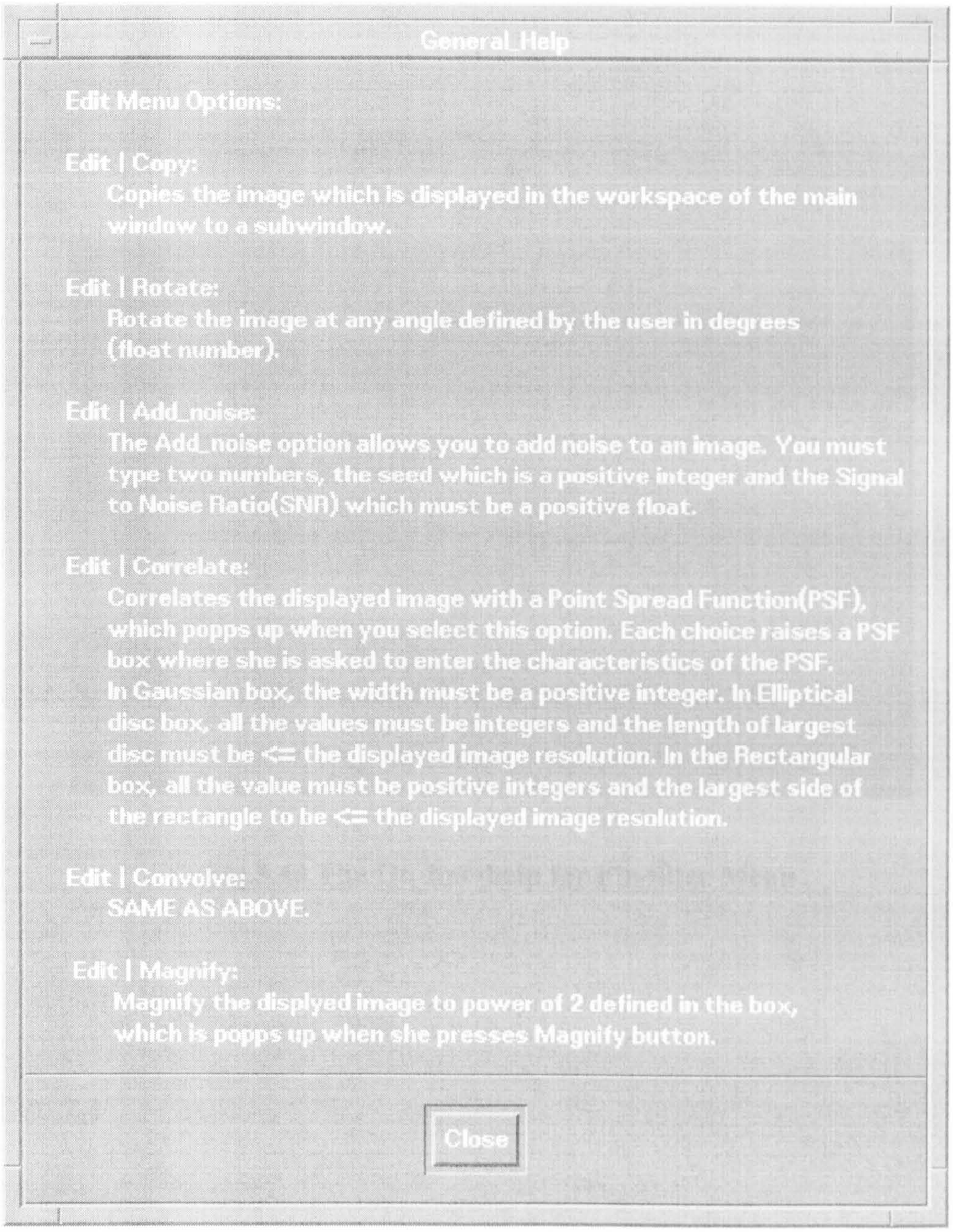


Fig 5.43 The On-line-help for Edit-Menu

C.3.15.4. Fir-Filter Menu Commands

As above. **(Fig 5.44)**

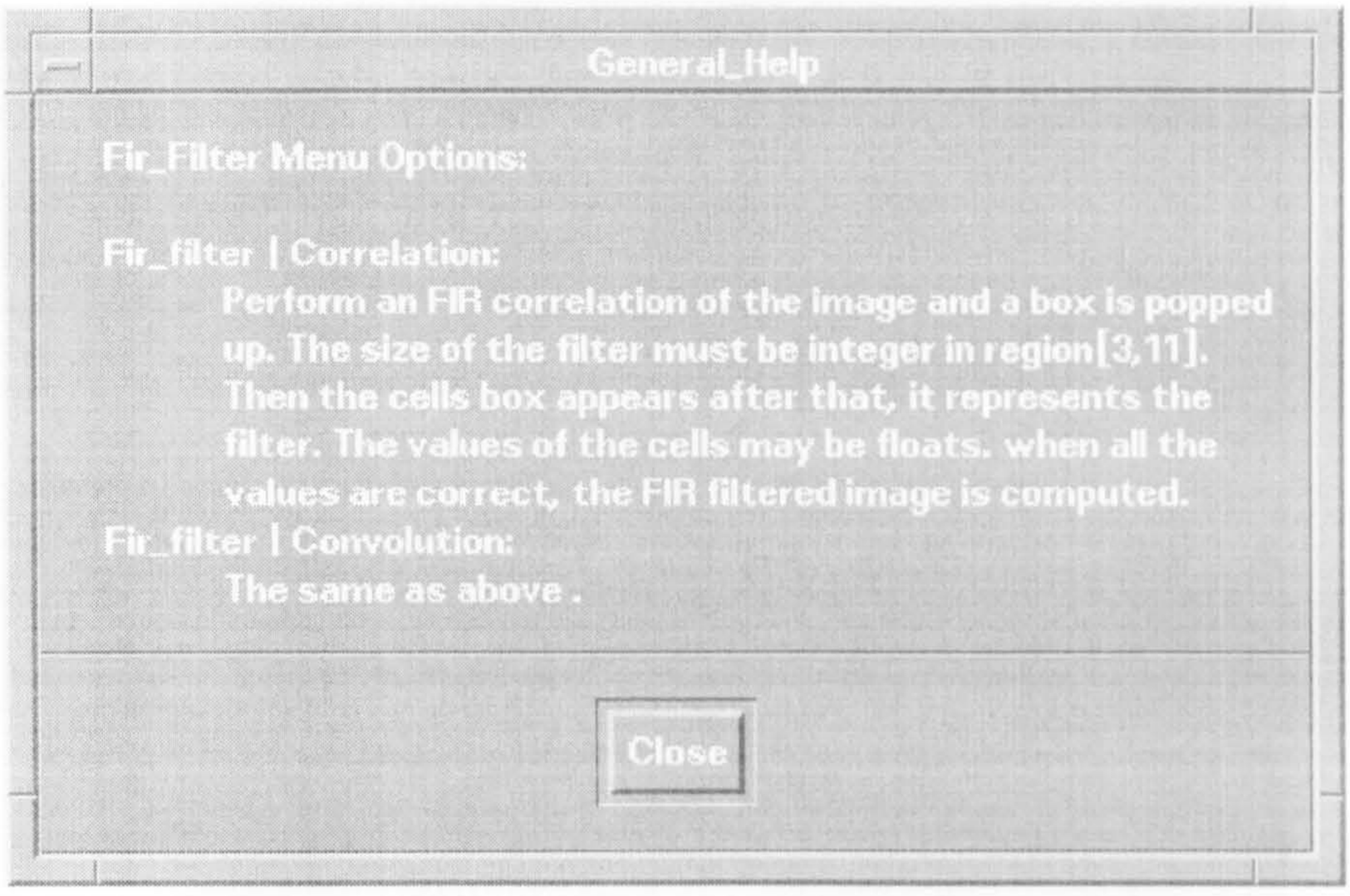


Fig 5.44 The On-line-help for Fir-filter Menu

C.3.15.5 Spectrum Menu Commands

As above. **(Fig 5.45)**

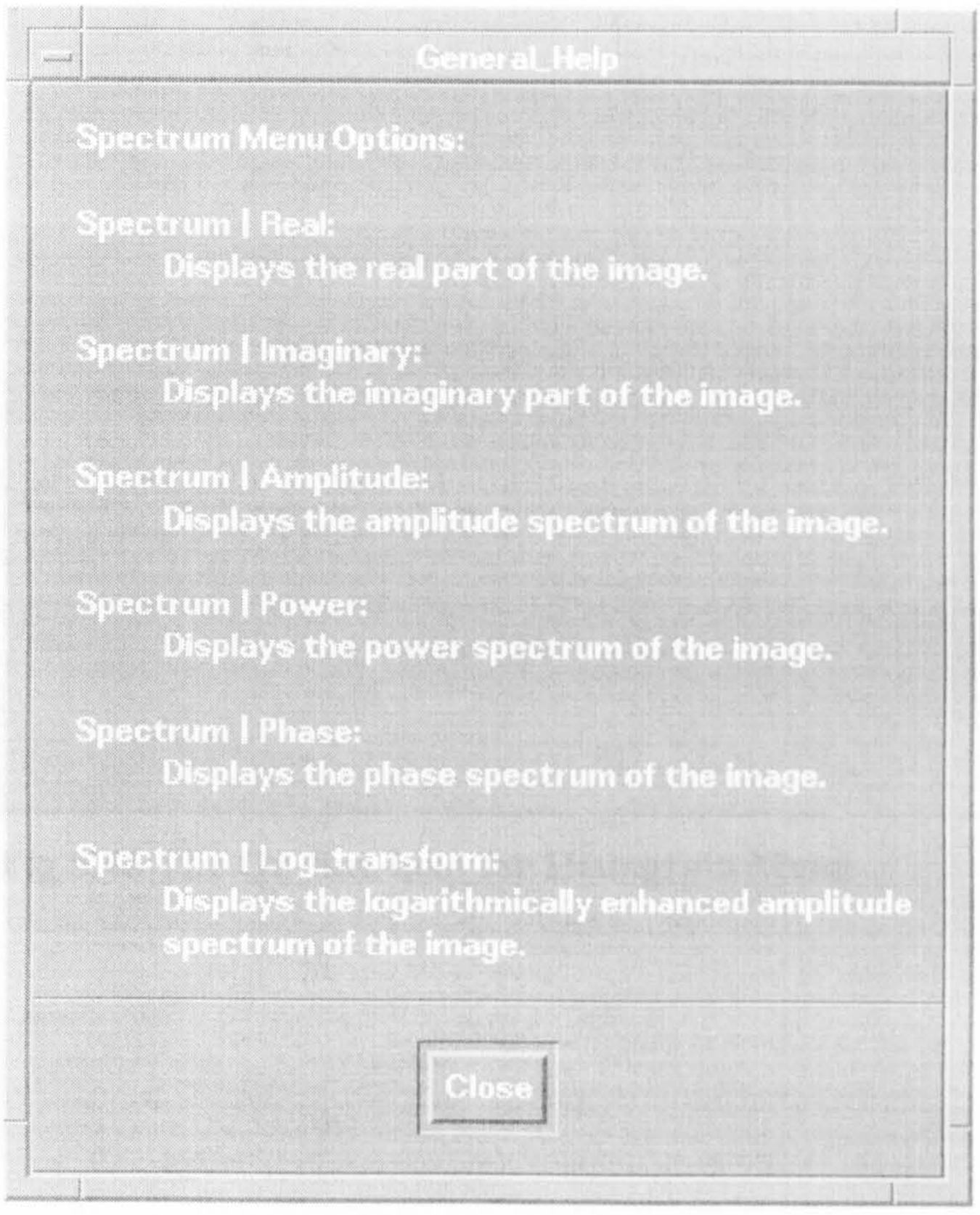


Fig 5.45 The On-line-help for Spectrum Menu

C.3.15.6. Histogram-Menu Commands

As above. **(Fig 5.46)**

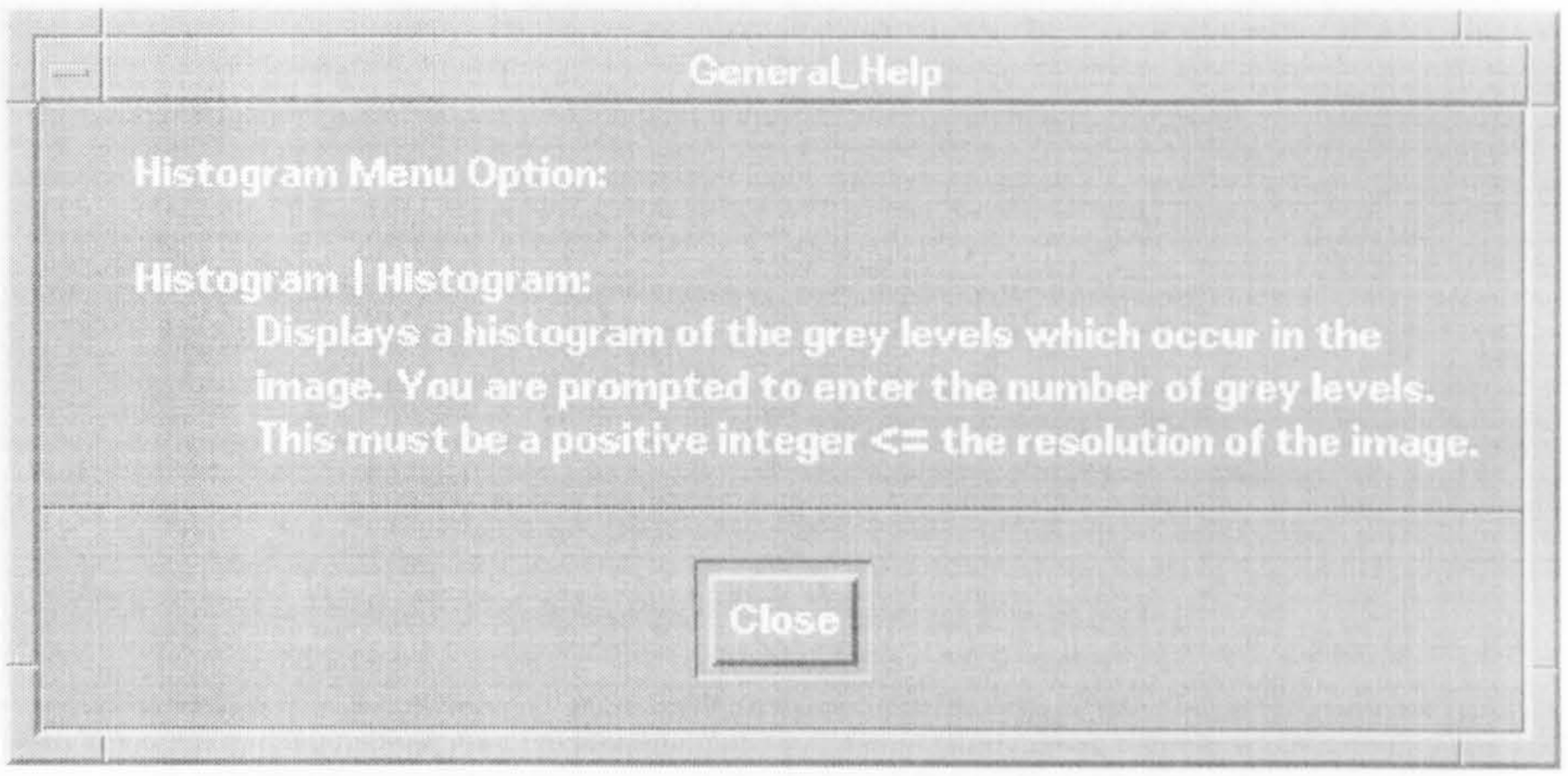


Fig 5.46 The On-line-help for Histogram Menu

C.3.15.7. Restoration-Menu Commands

As above. **(Fig 5.47)**

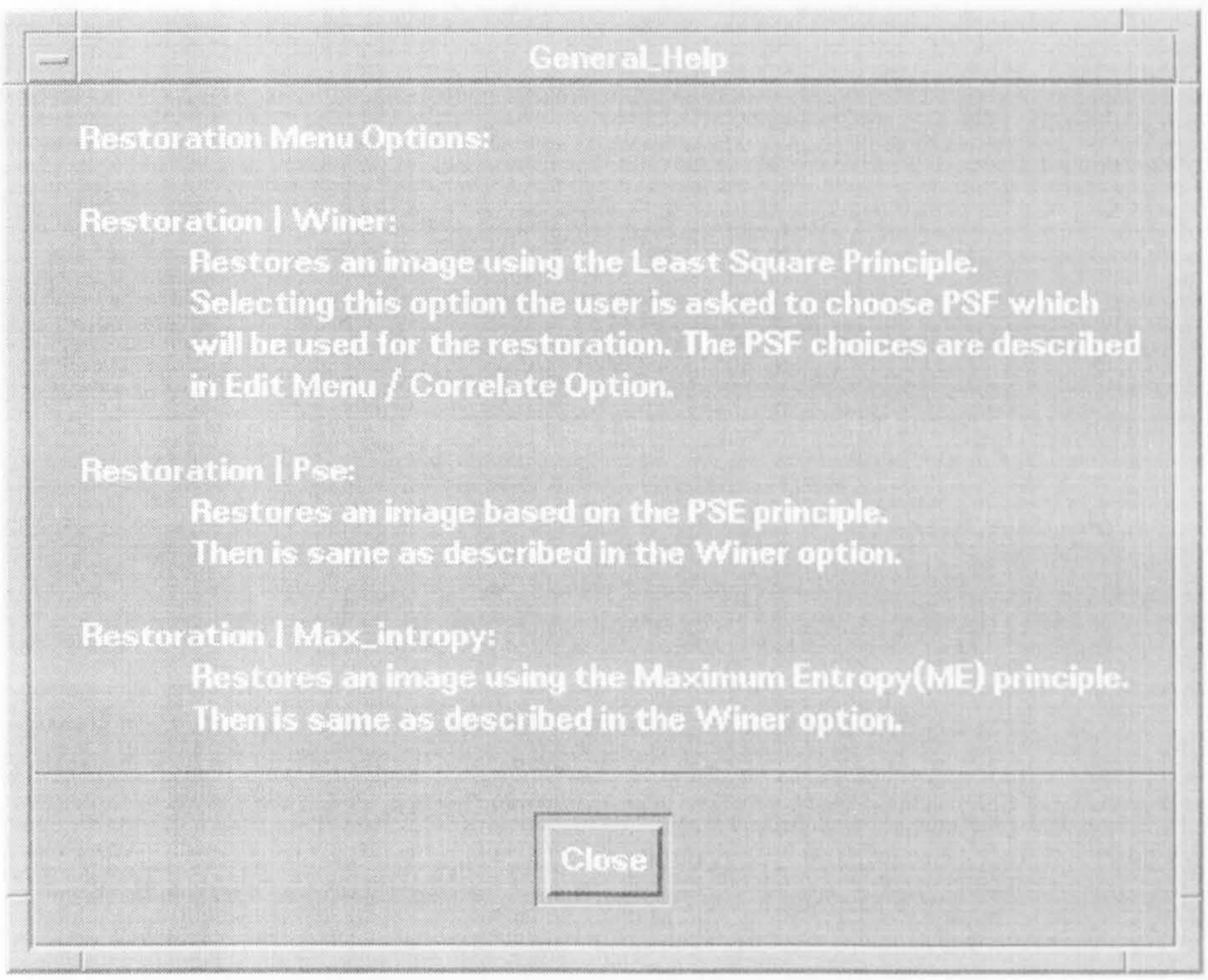


Fig 5.47 The On-line-help for Restoration Menu

C.3.15.5. Ann-Menu Commands

As above. **(Fig 5.48)**

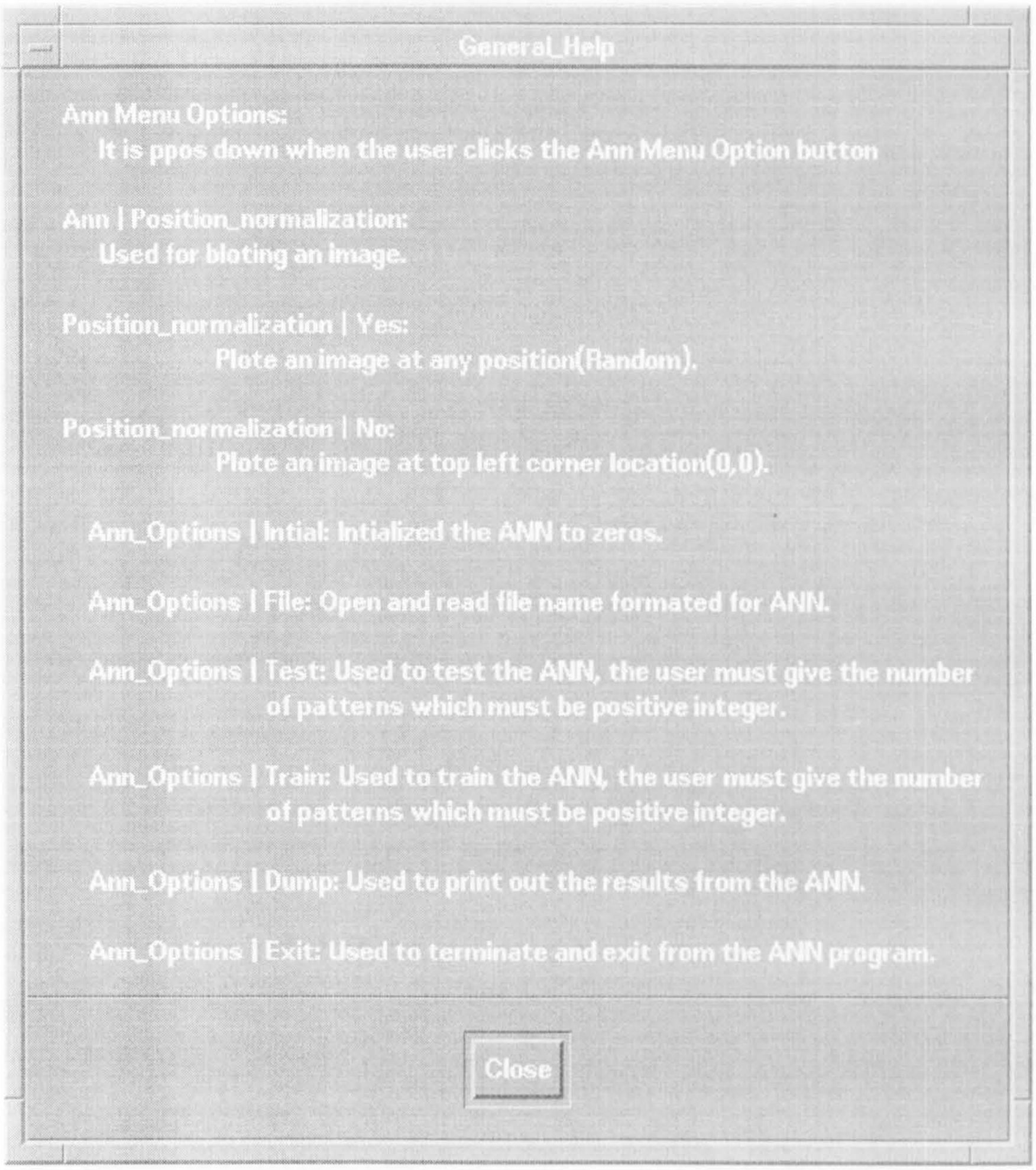


Fig 5.48 The On-line-help for Ann Menu

C.3.15.9. Enhancement-Menu Commands

As above. **(Fig 5.49)**

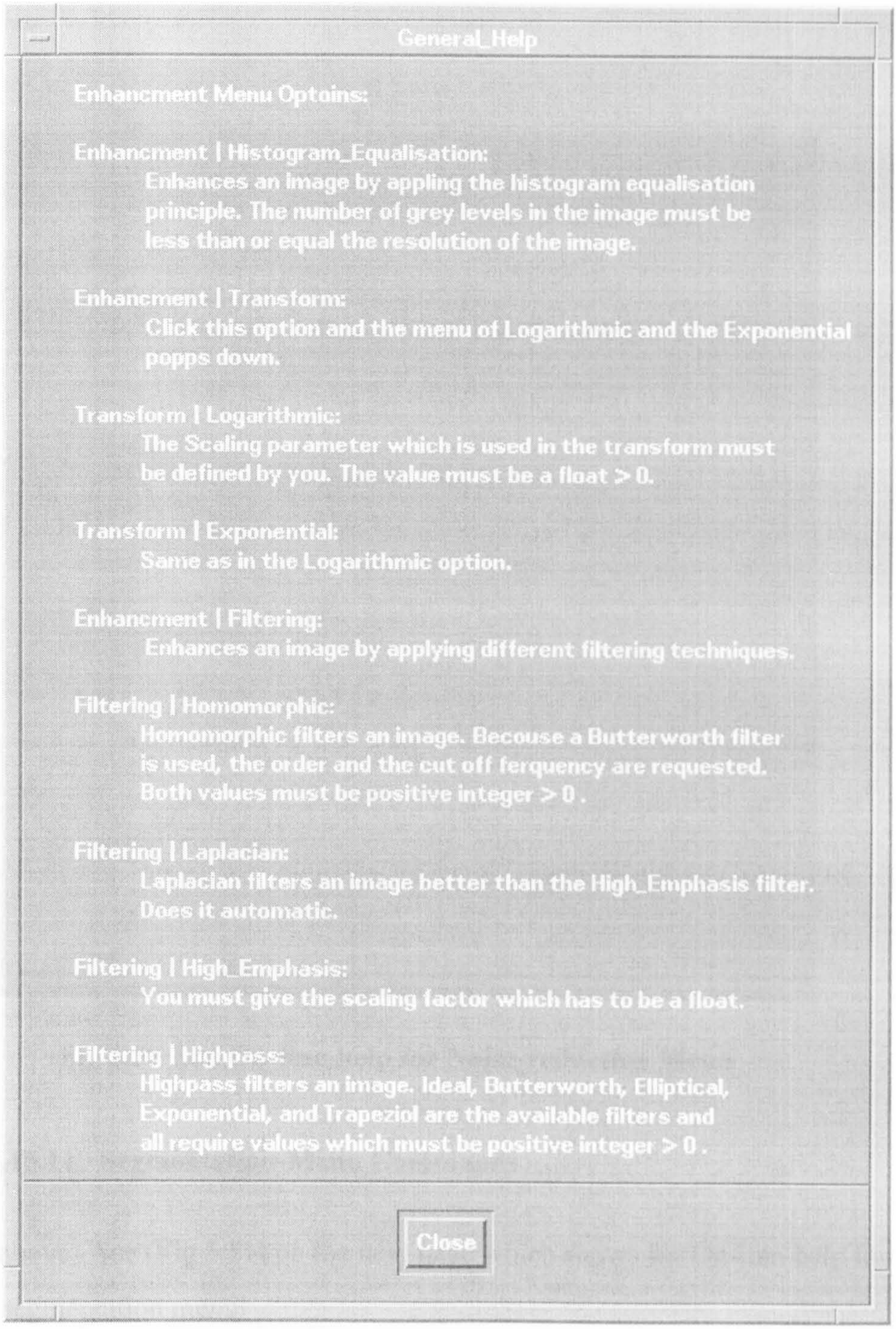


Fig 5.49 The On-line-help for Enhancement Menu

C.3.15.10. Noise-reduction Menu Commands

As above. **(Fig 5.50)**

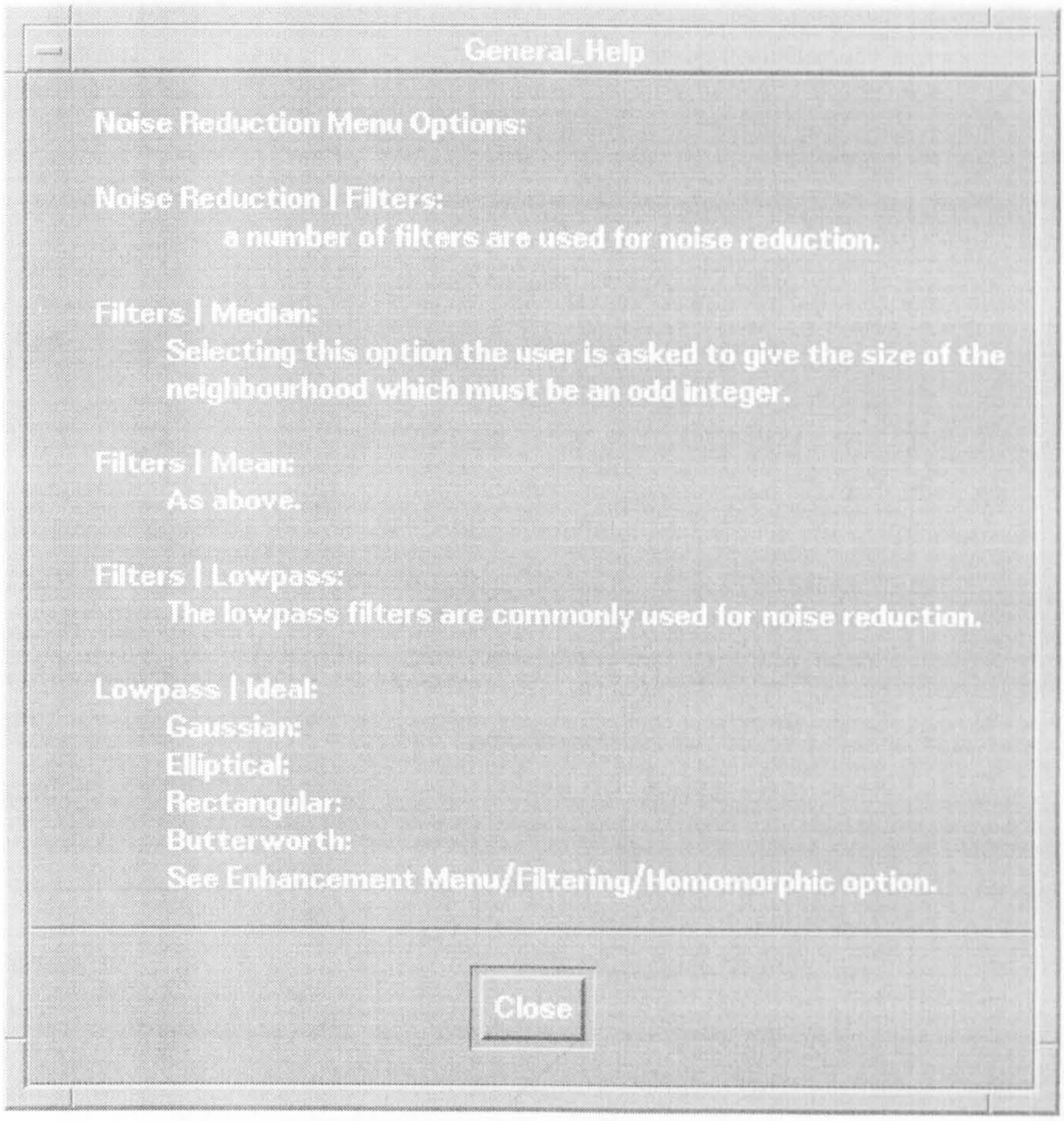


Fig 5.50 The On-line-help for Noise-reduction Menu

C.3.15.11. Segmentation-Menu Commands

As above. See **(Fig 5.51)** on the next page which shows the On-line-help for the segmentation menu.

General_Help

Segmentation Menu Options:
Appears when the user clicks the Segmentation or F10.

Segmentation | Threshold:
Used for segmenting an image into region of similarity.

Threshold | Single Band:
You must provide the threshold value, a positive float.

Threshold | Semi Threshold:
You must give the semi_Threshold which is a positive float < 1.

Threshold | Multi Band:

The user is prompted to select from a toggle button box the thresholds number, there are six. Any of thes selection expected to type the thresholds values which must be positive float < 1.

Segmentation | Edge Detection:
Segments an image into regions of discontinuity.

Edge Detection | First Order:
First order detector is provided for edge detection.

Edge Detection | Sobel:
Using the Sobel, you must select the direction of the edge detection from toggle box.

Edge Detection | Prewitt:
Operates as the Sobel except that the prewitt operater is used

Edge Detection | Roberts:
Roberts operator is provided for edge detection.

Edge Detection | Orthogonal:
You must select the threshold for Orthogonal edge detection.

Edge Detection | Atan:
Edge detector using Tan Inverse operator.

Edge Detection | Second Order:
Second order detectors are provided for edge detection.

Edge Detection | Edge Enhanc ANN:
Edge detetor is provided using Aritficial Neural Network.

Close

C.3.15.12. Radon-Menu Commands

As above. **(Fig 5.52)**

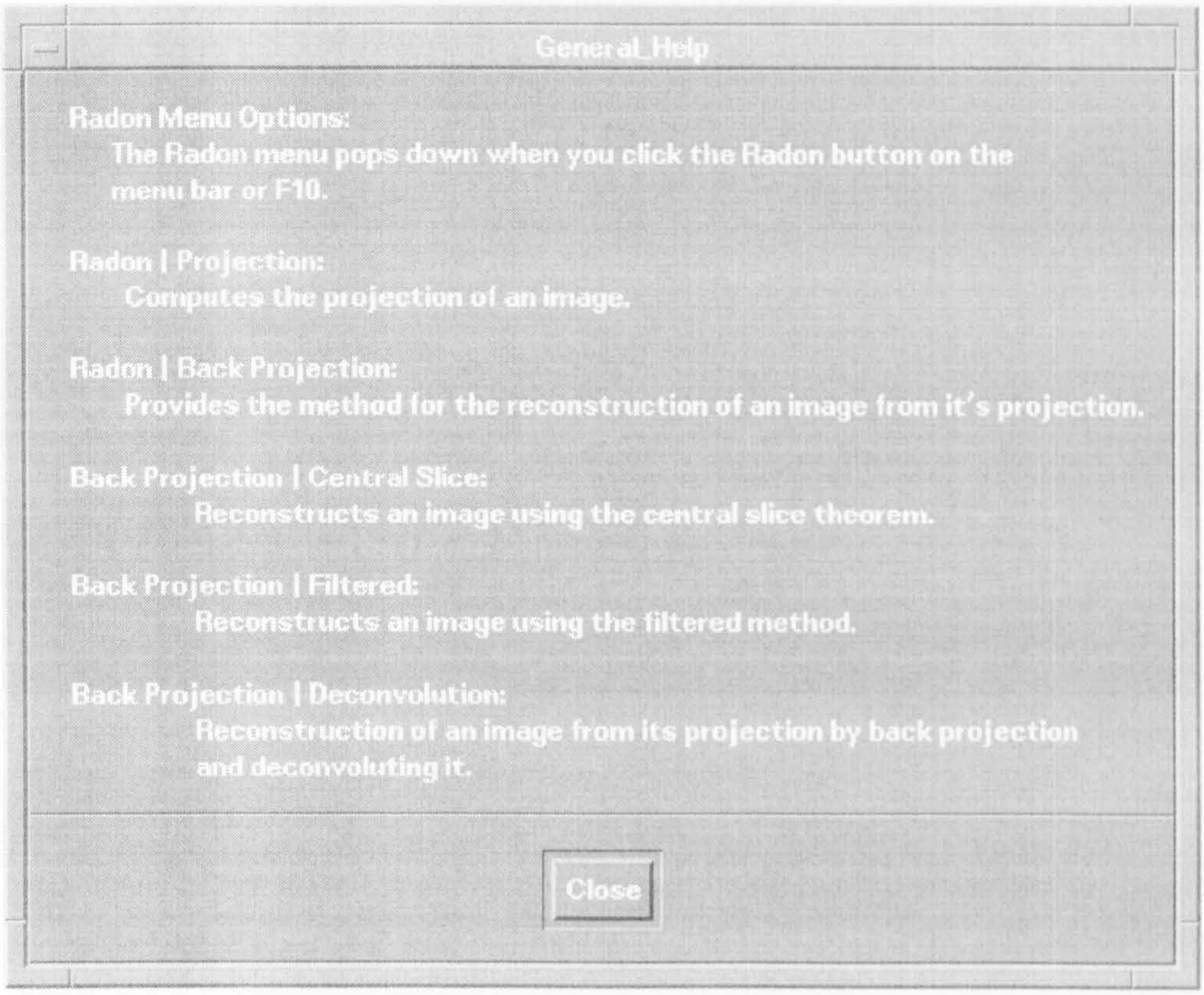


Fig 5.52 The On-line-help for Radon Menu

C.3.15.13. Fractal-Menu Commands

As above. **(Fig 5.53)**

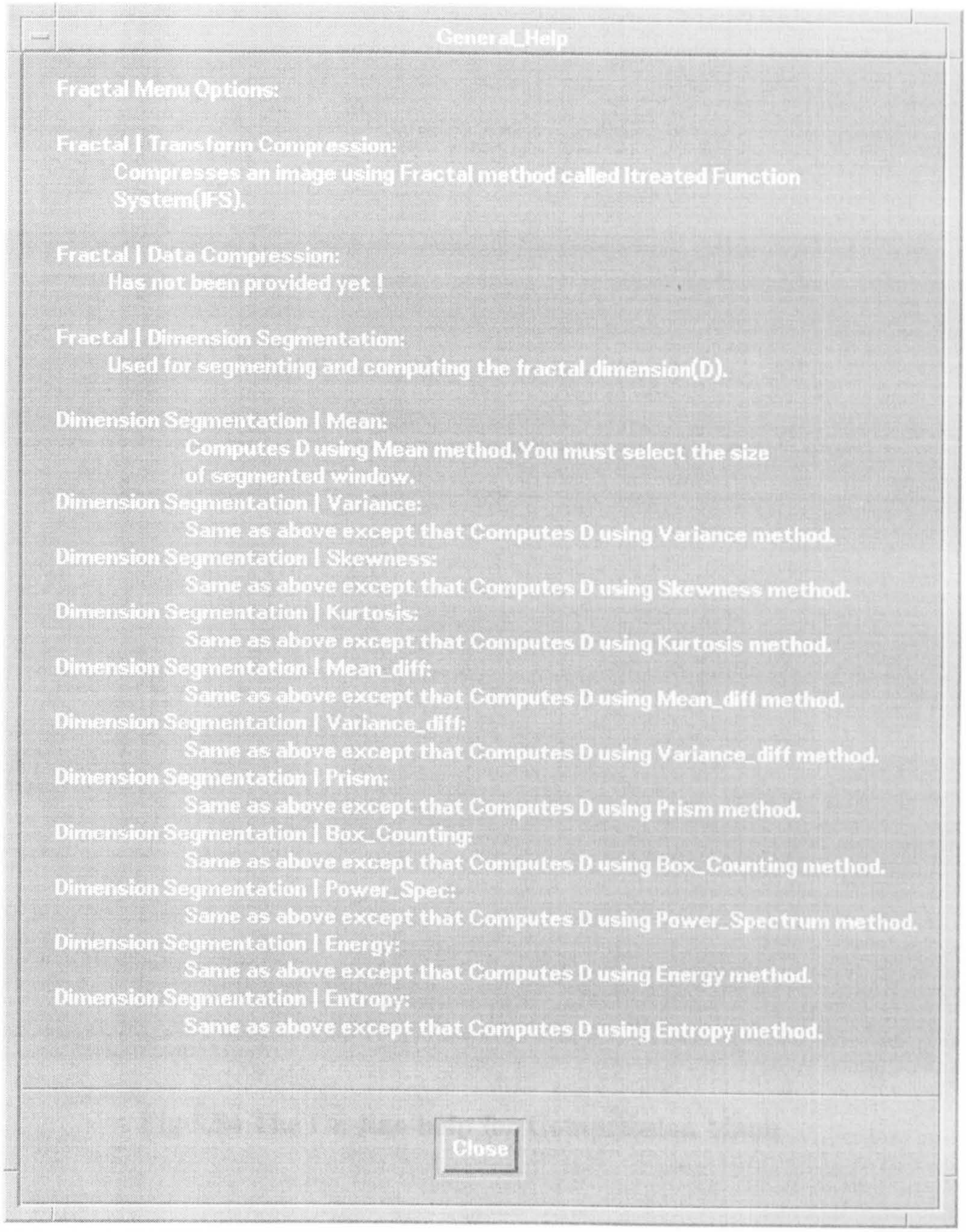


Fig 5.53 The On-line-help for Fractal Menu

C.3.15.14. Compression-Menu Commands

As above. **(Fig 5.54)**

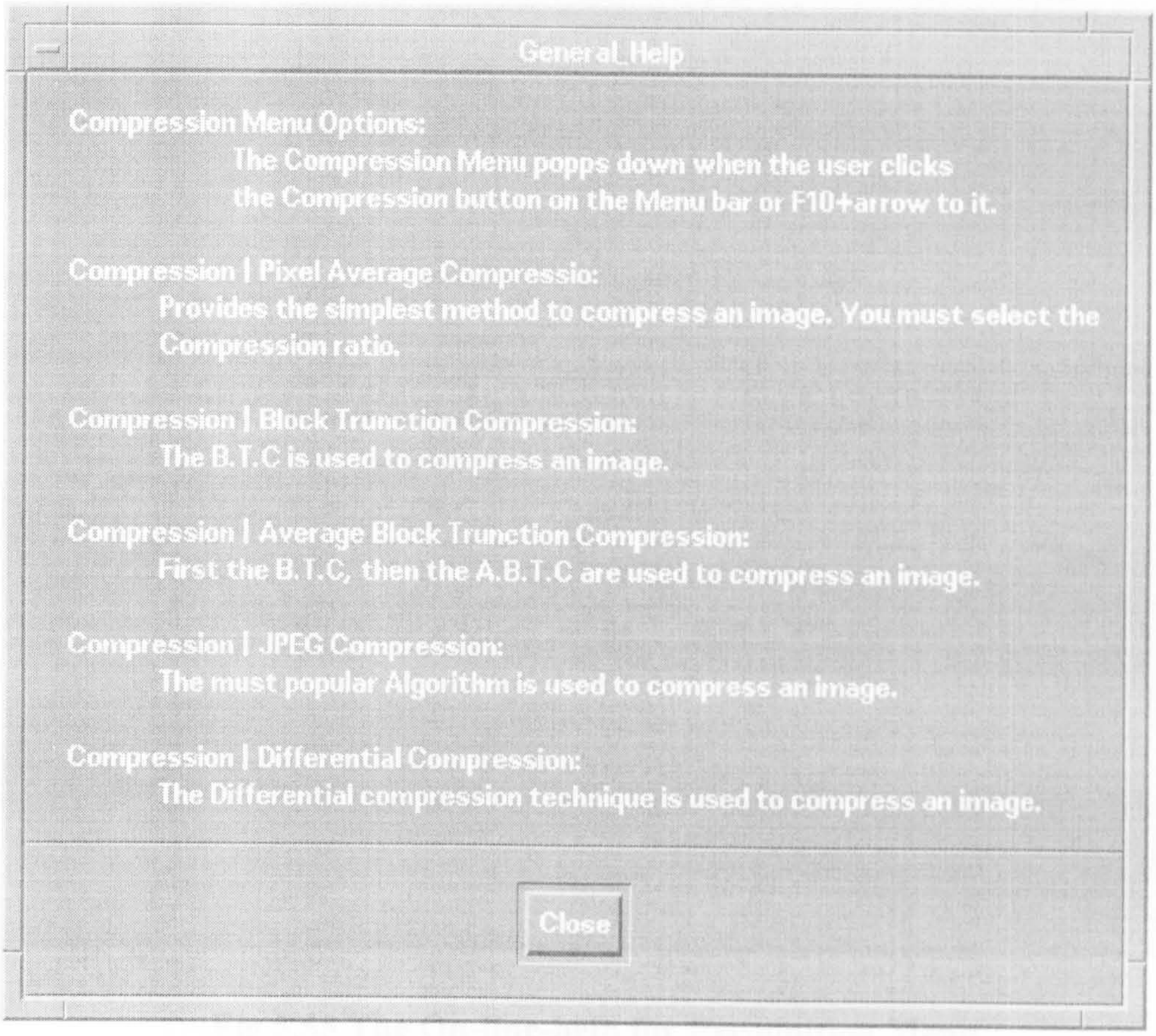


Fig 5.54 The On-line-help for Compression Menu

C.3.15.15. Recognition-Menu Commands

As above. **(Fig 5.55)**

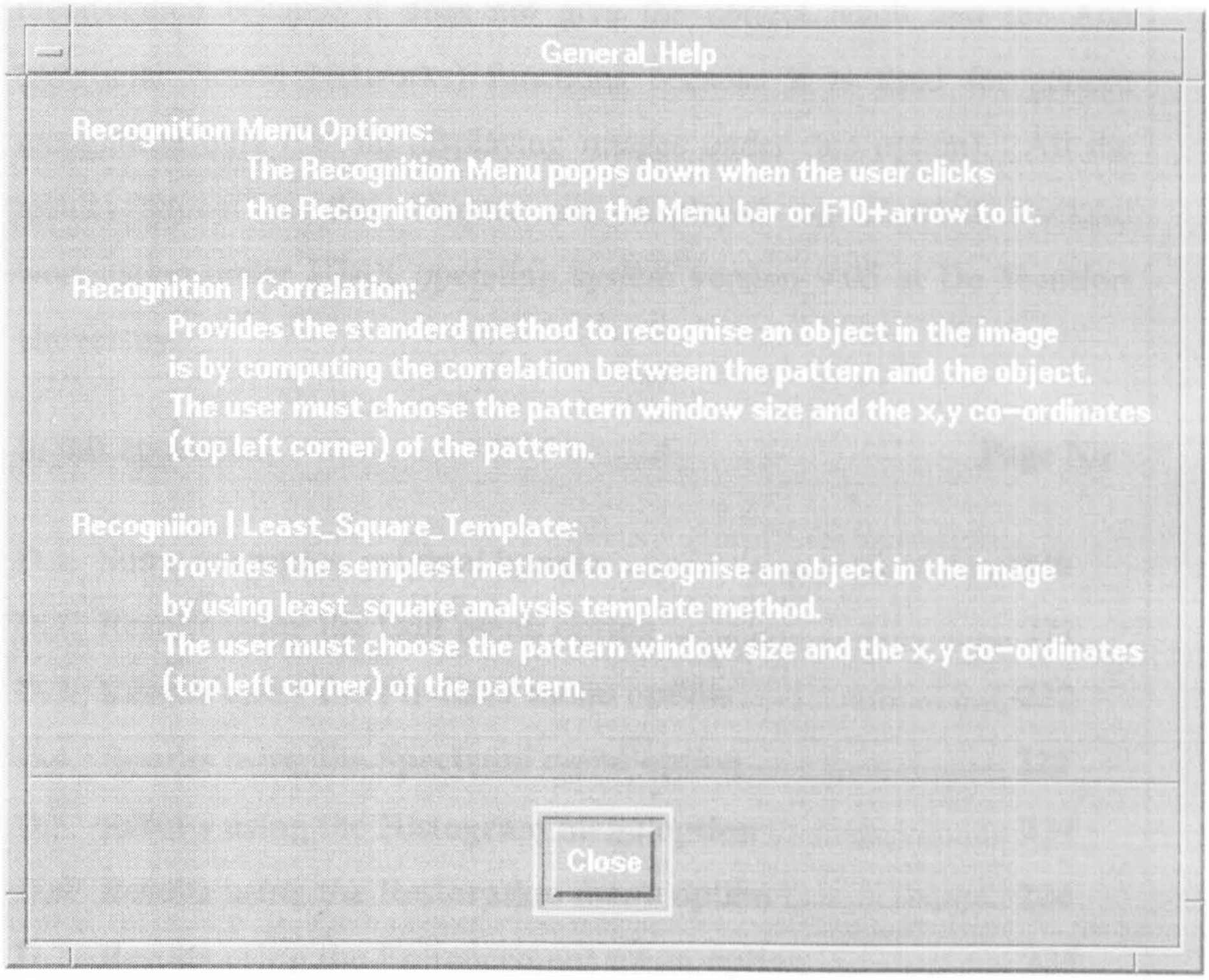


Fig 5.55 The On-line-help for Recognition Menu

Appendix D

Supplementary Results

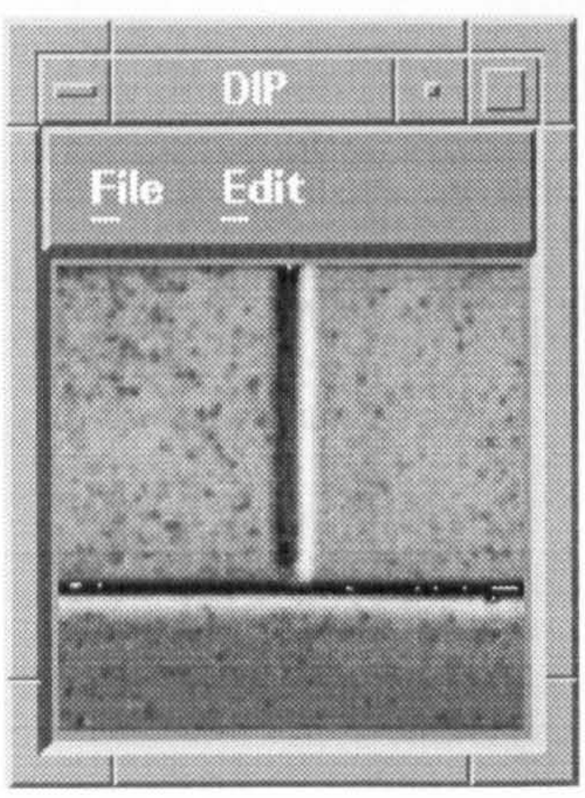
In this chapter we represent the output results of the DIP package functions except the Filtered Back-projection which has to be reconsidered because it does not give the correct result and the Ann (Artificial Neural Networks) functions because it is used for pattern recognition only (i.e. no displaying images under this option). All the results shown in this chapter are displayed on Hewlett Packard workstation under UNIX operating system version 9.05 at De Montfort University.

In this appendix: **Page No**

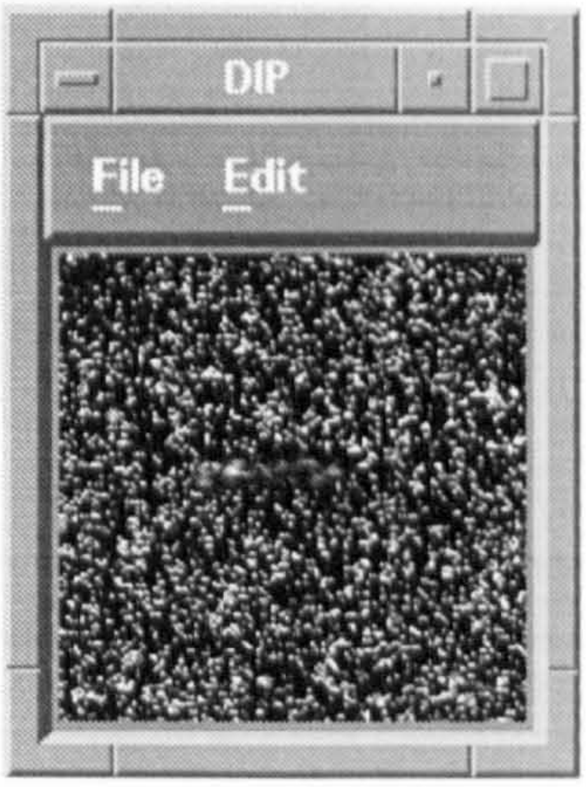
D.1. Some examples, original images

Lena original image

Mona original image



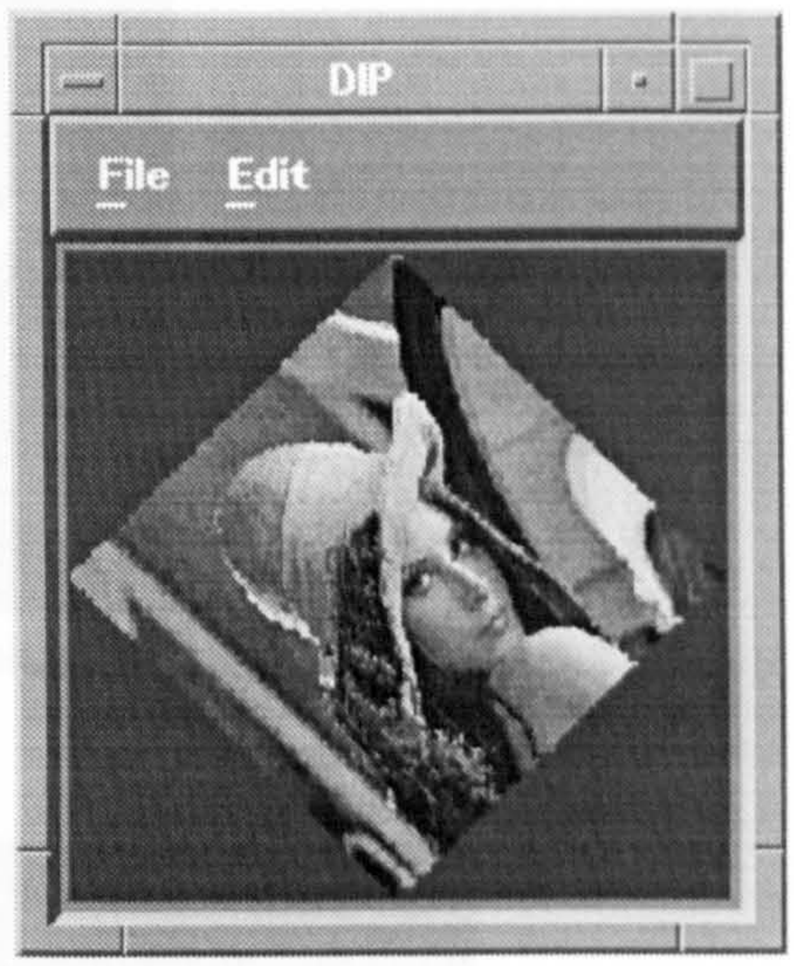
Original image



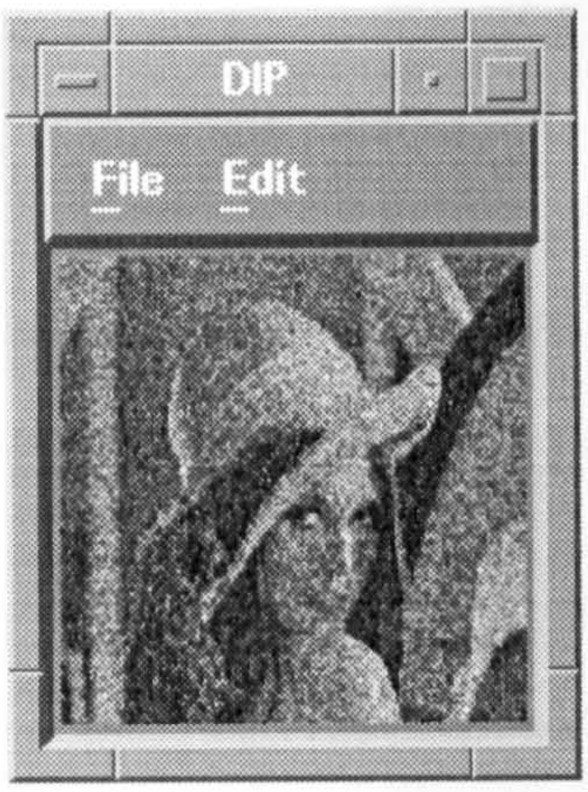
Original Ship Image

D.2. Results using the Edit menu option

Copy: Lena



Lena: Rotated by



45°

Lena: Add noise

Lena: Correlated

Lena: Convoluted

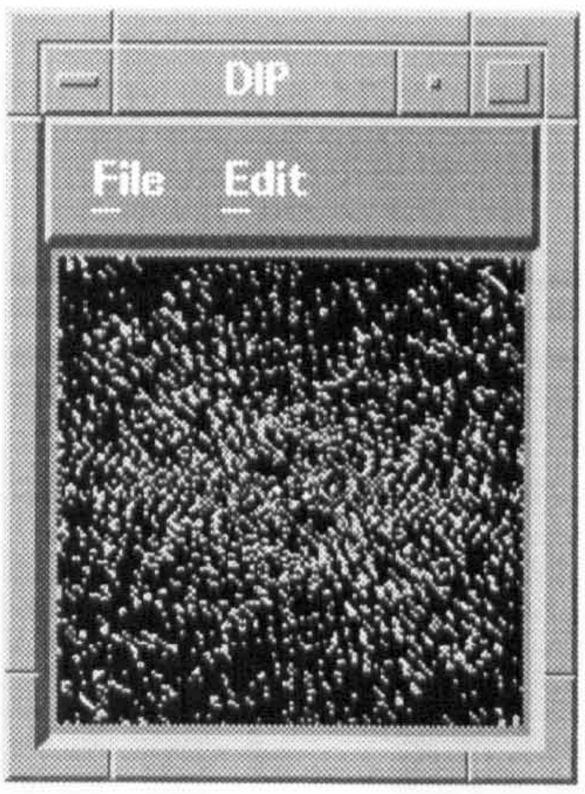
Lena: Magnified (vector = 2)

D.3. Results using Fir-filter menu option

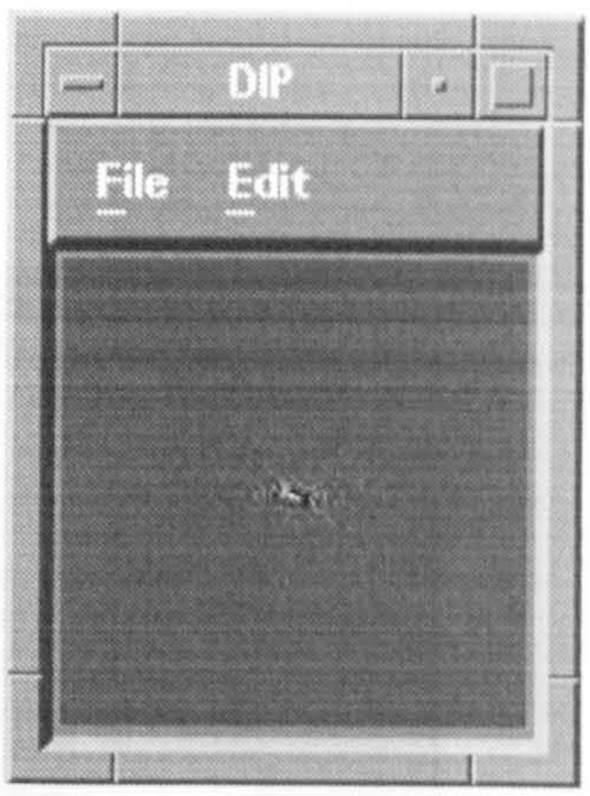
Lena: Correlated

Lena: Convoluted

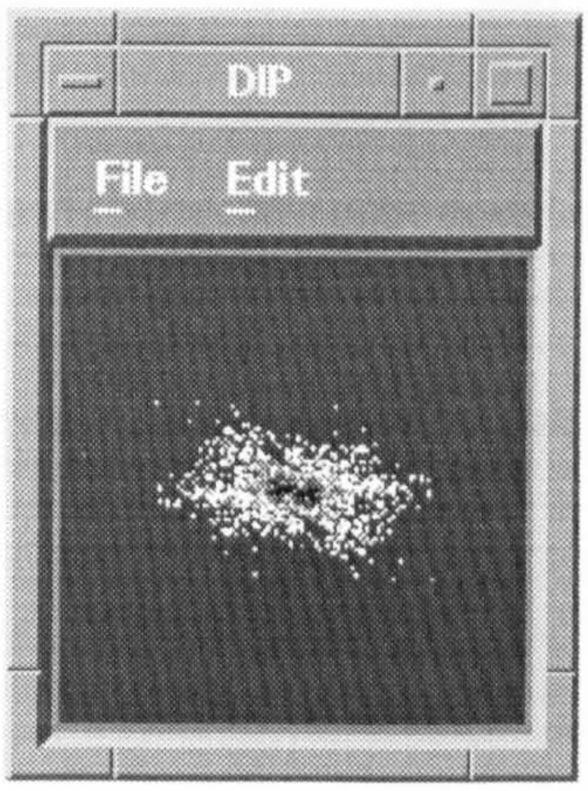
D.4. Results using the Spectrum menu option

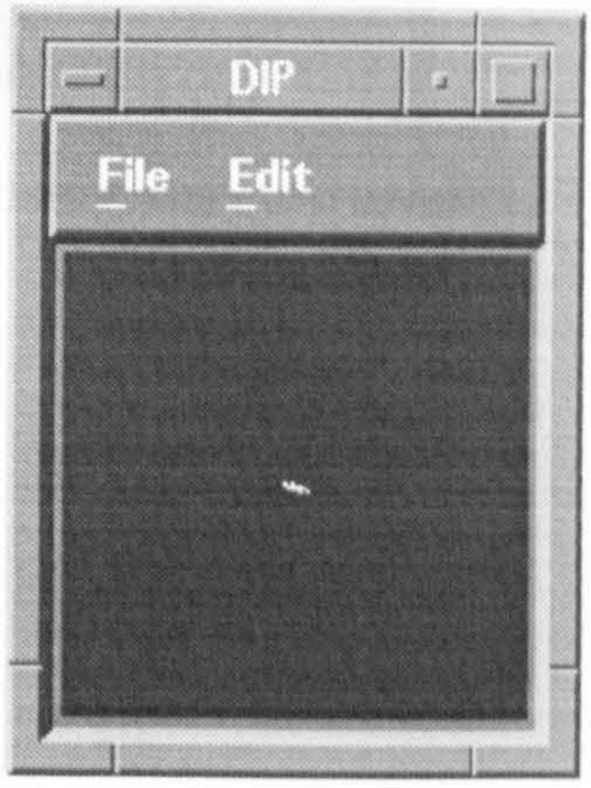


Lena: Real part

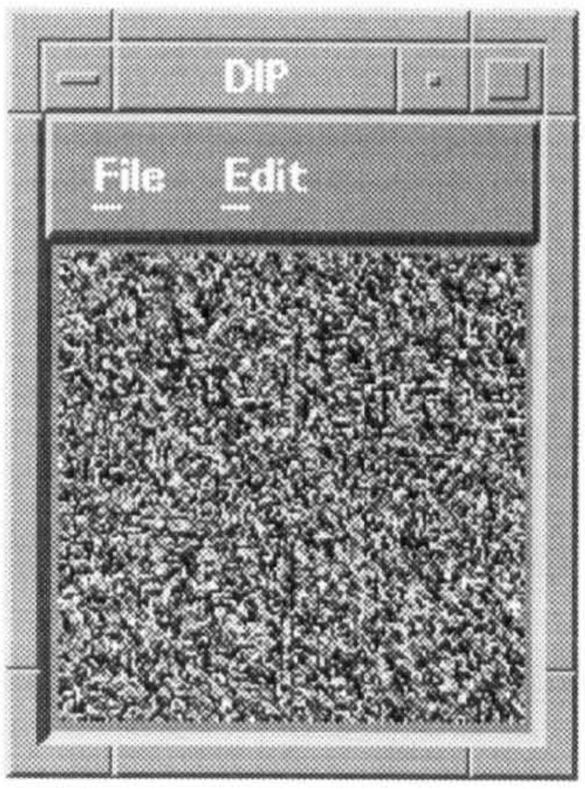


Lena: Imaginery part

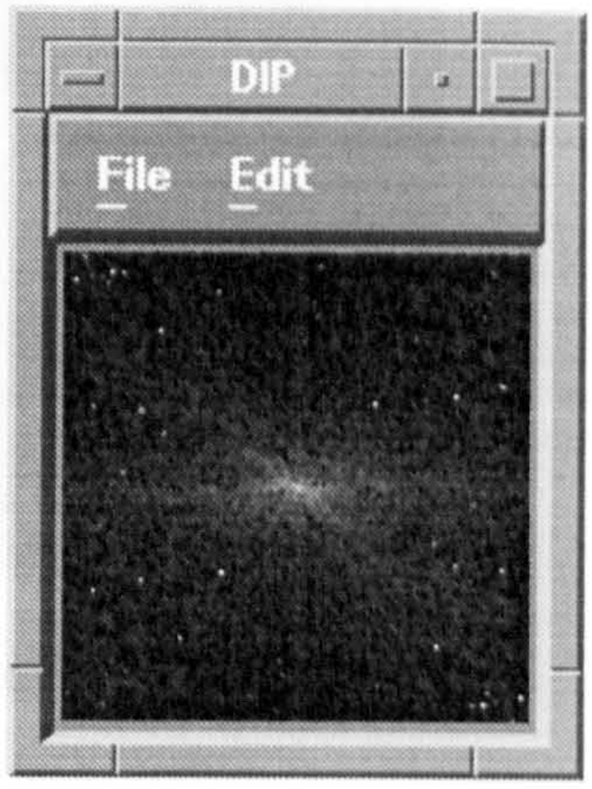




Lena: Amplitude part



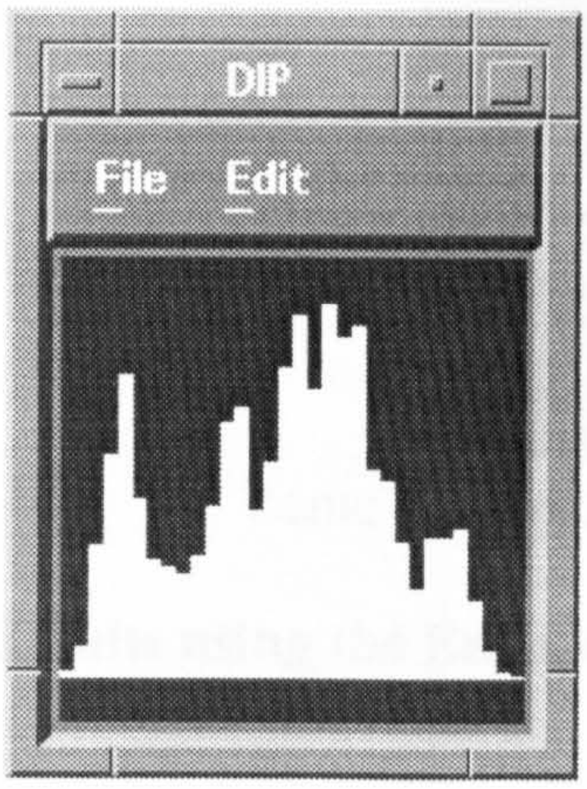
Lena: Power Spectrum

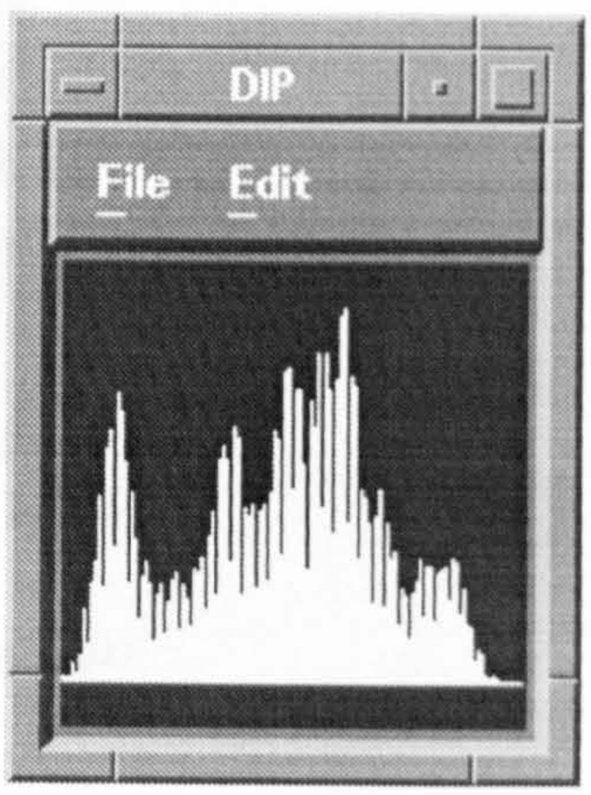


Lena: The Phase part

Lena: The Log. part

D.5. Results using Histogram menu optin

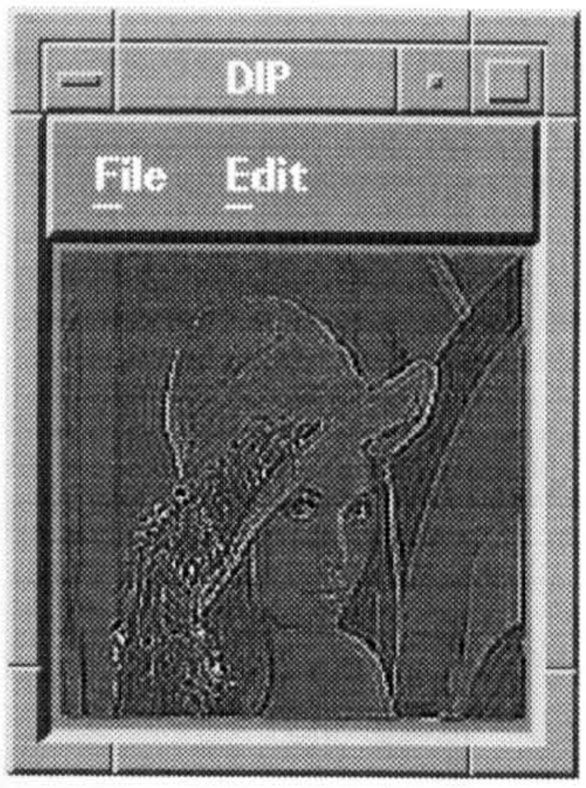




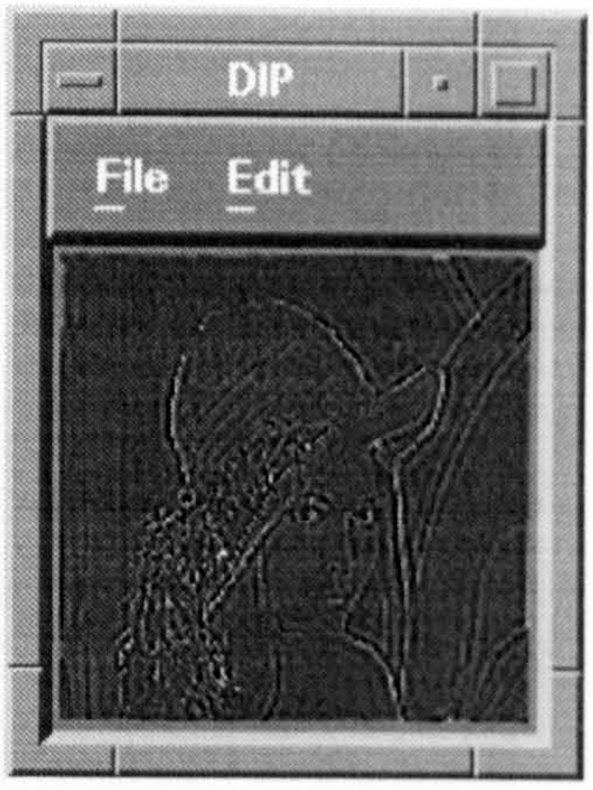
Lena: Histogram(grey level=32)

Lena: Histogram(grey level=127)

D.6 Results using Restoration menu option



Lena: Restored /Winer/Gaussian



Lena: Restored /Pse/Gaussian

Lena: Restored/Max-entorpy/Gaussian

D.7. Results using the Enhancement menu option

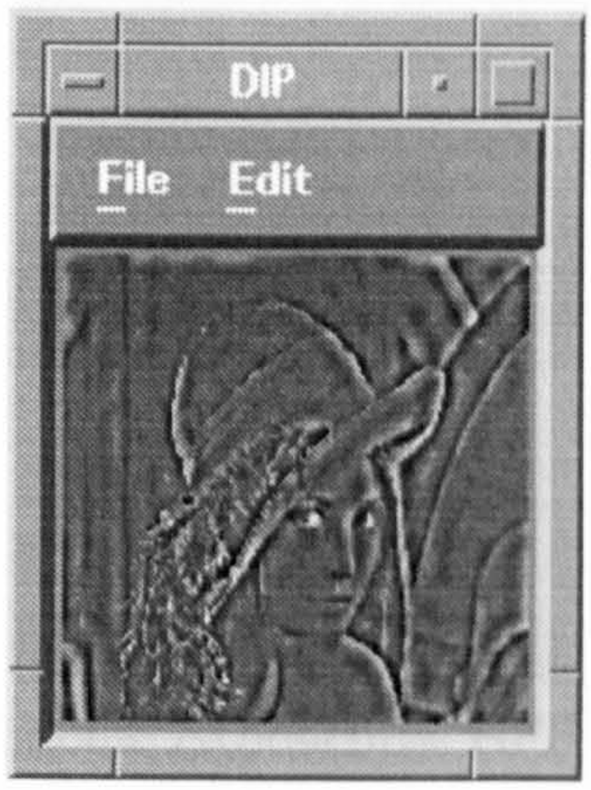
Lena: Histogram equalisation

Lena:

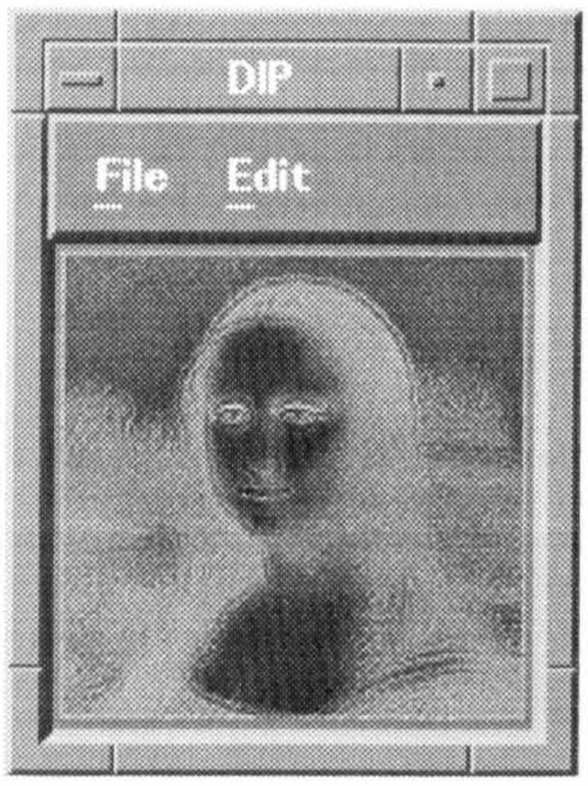
Histogram equalisation
(greylevel=8)

Lena: Transform/Logarithm

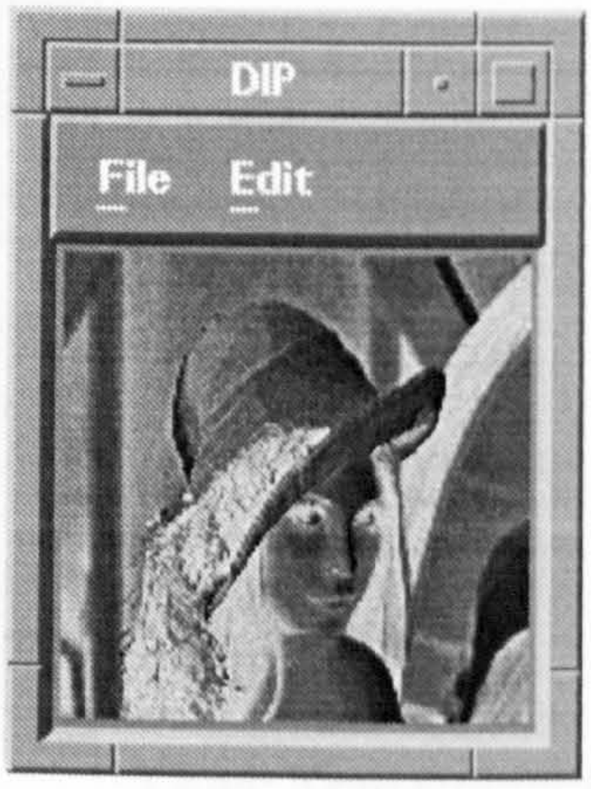
(Grey level=100)



Lena: Transform/Exponent

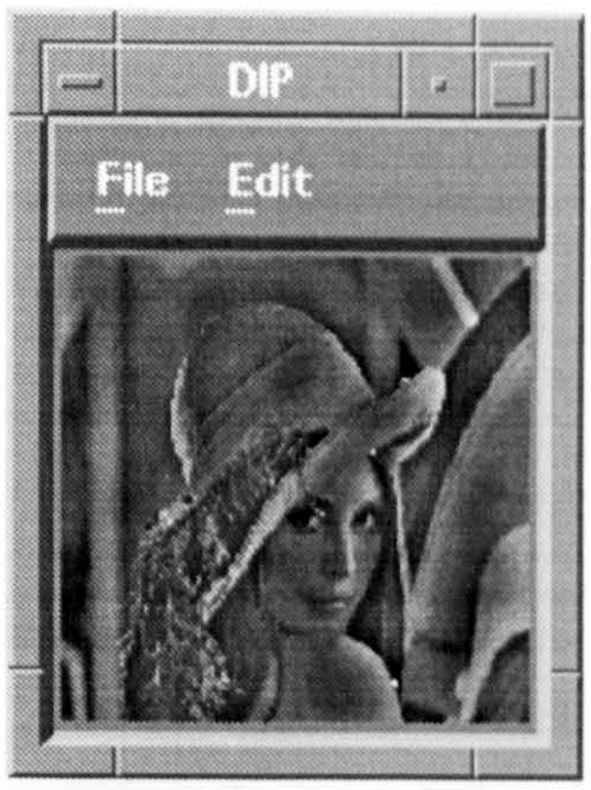


Mona: Laplacian

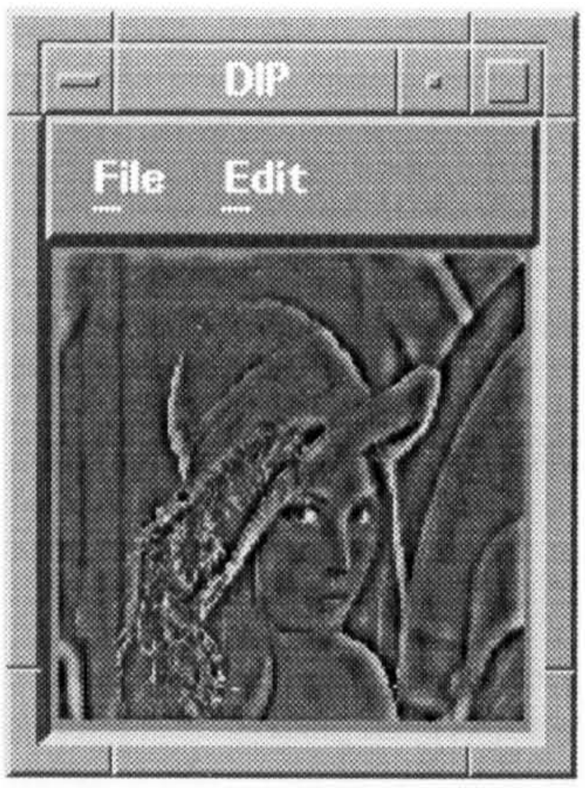


Lena: Laplacian

Lena: High-emphasis

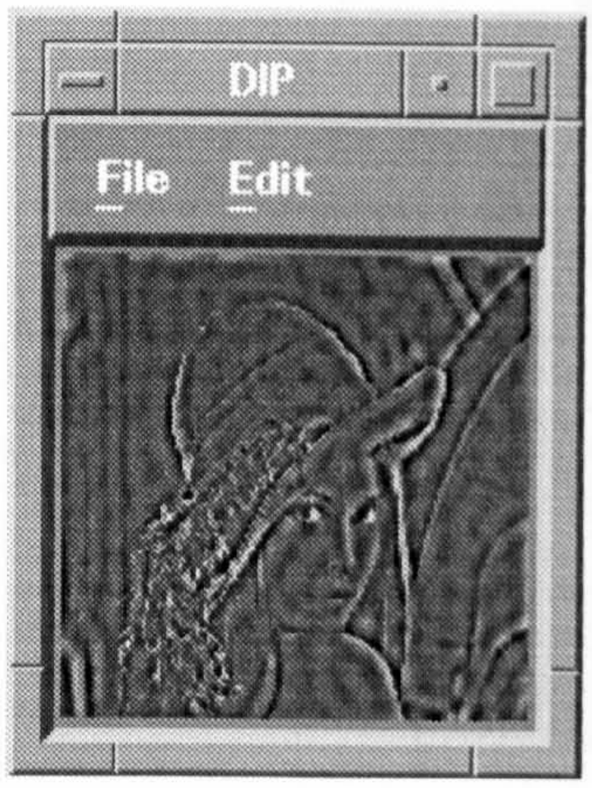


Lena: Ideal HPF



Lena: Butterworth HPF

Lena: Exponential HPF

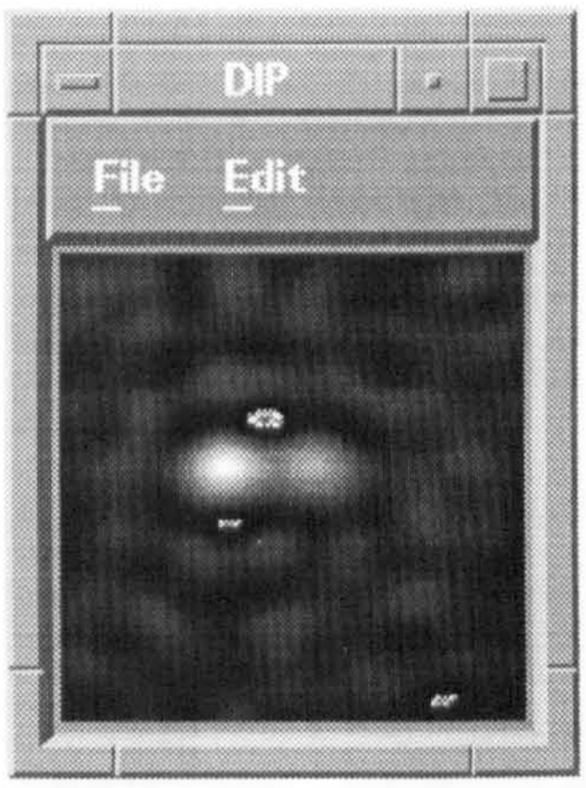


Lena: Trapezoidal HPF

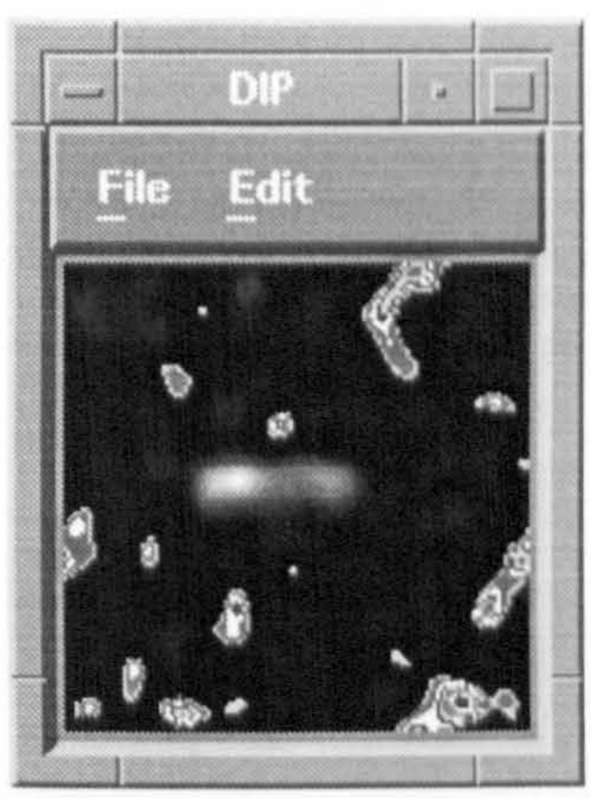
D.8. Results using the Noise-reduction menu option

Lena: Noise/Median Filter

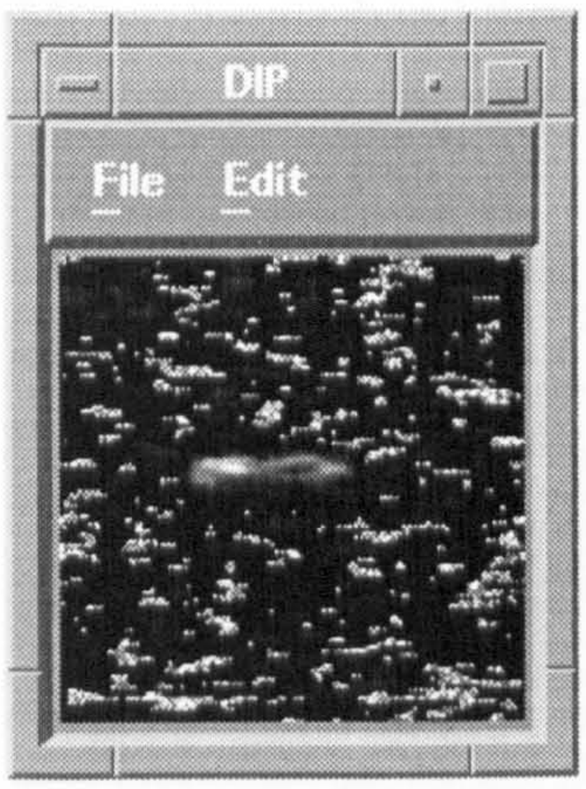
Lena: Noise/Mean Filter



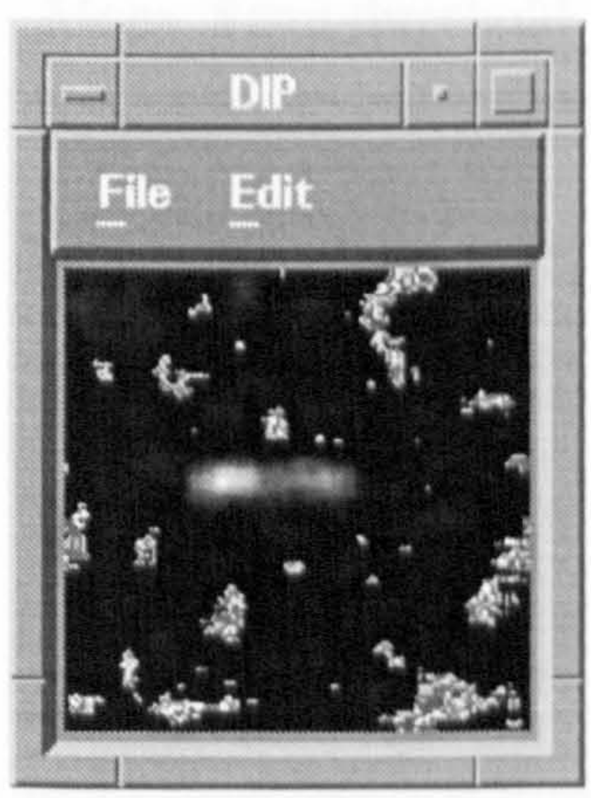
Ship: Noise/Ideal LPF



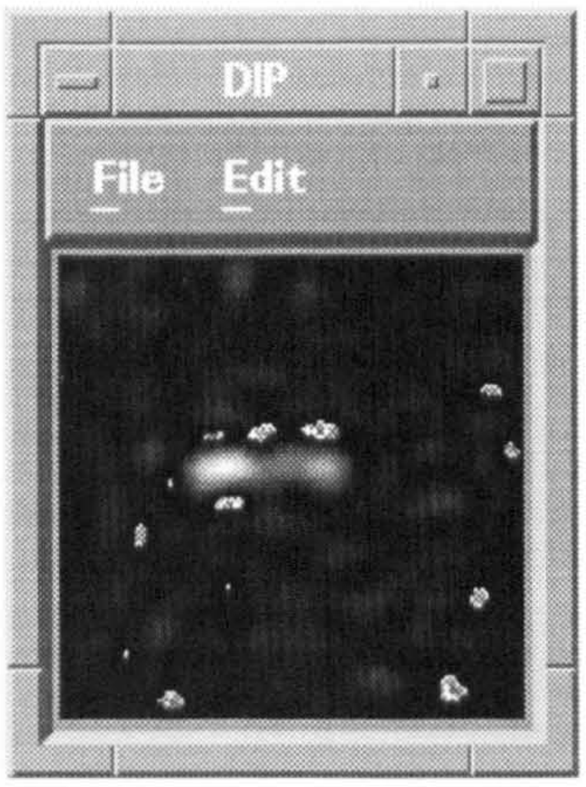
Ship: Noise/Gaussian LPF



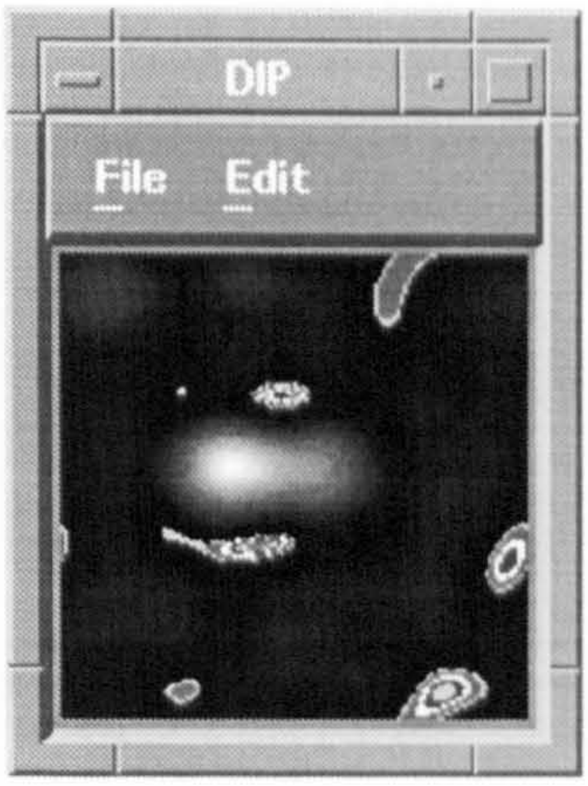
Ship: Elliptic LPF



Ship: Rectangular LPF



Ship: Butterworth LPF



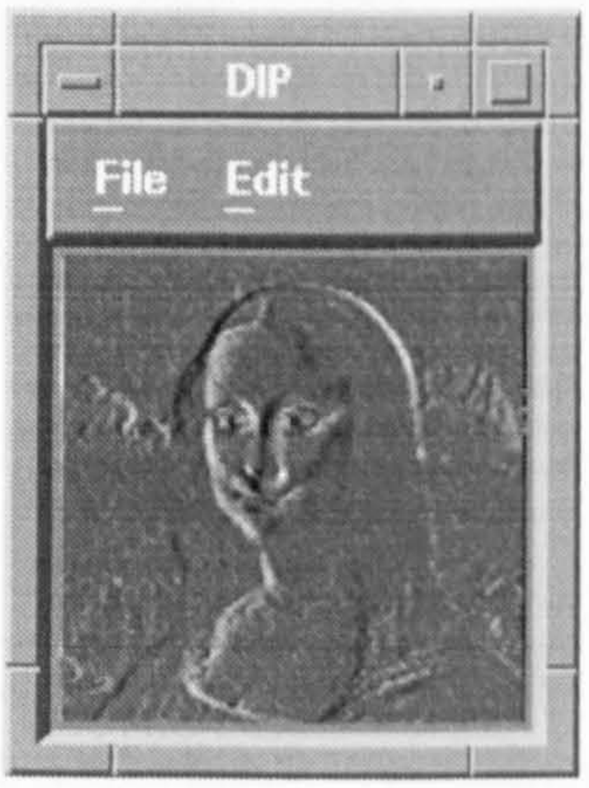
Ship: Trapezoidal LPF

D.9. Results using Segmentation Menu option

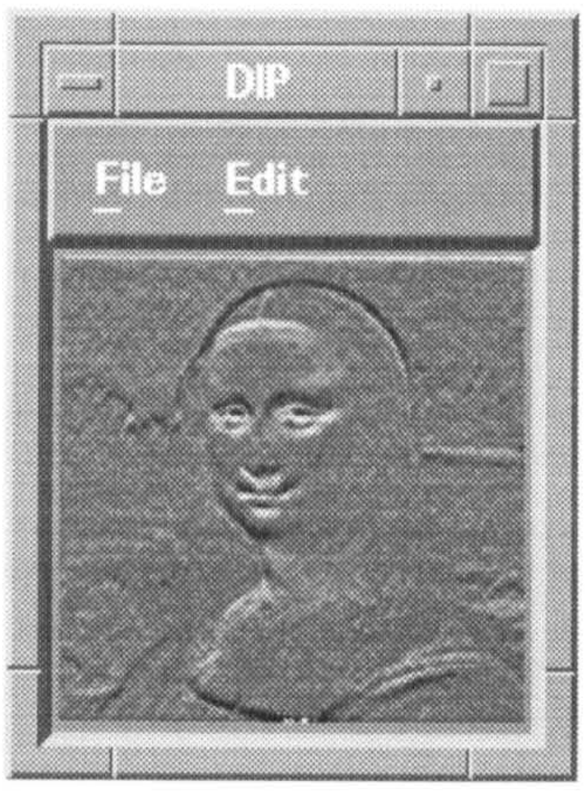
Mona: Single-band threshold

Mona: Semi-threshold

Mona: Multi-band threshold

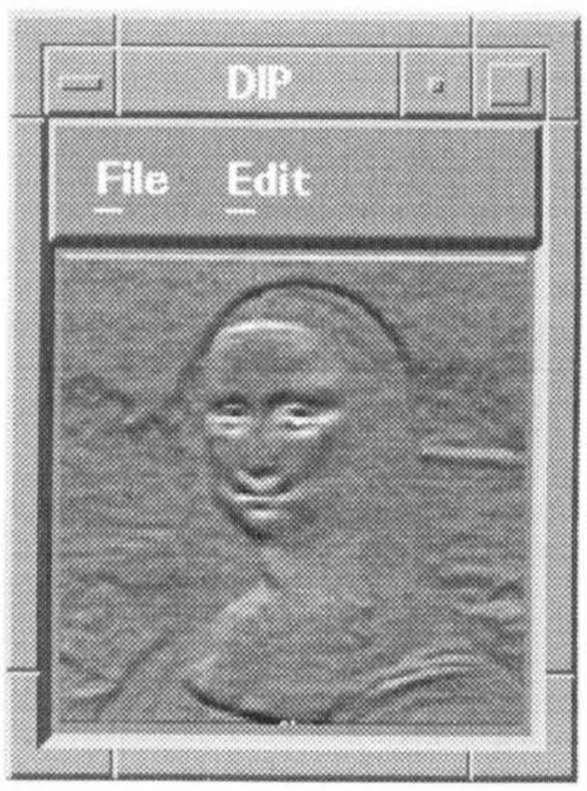


Mona: Sobel edge detection
(x - direction)

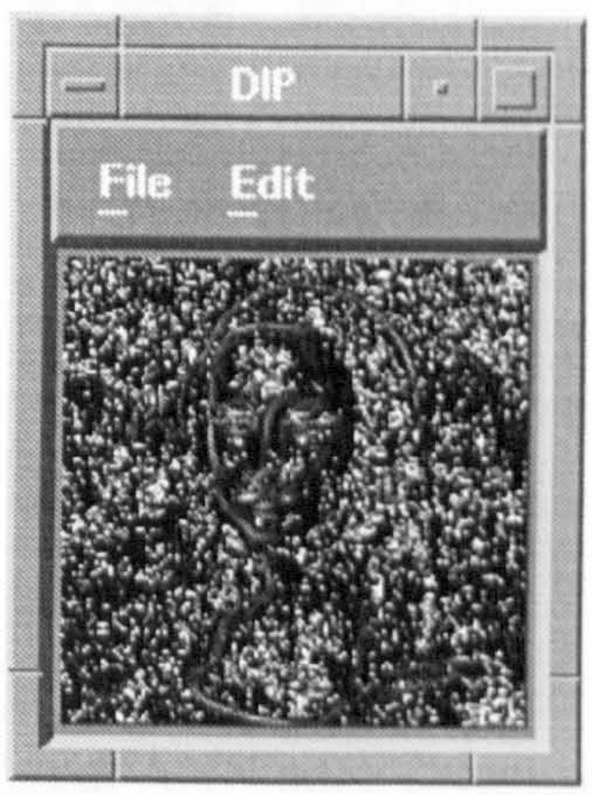


Mona: Sobel edge detection
(y - direction)

Mona: Prewitt edge detection
(x - direction)

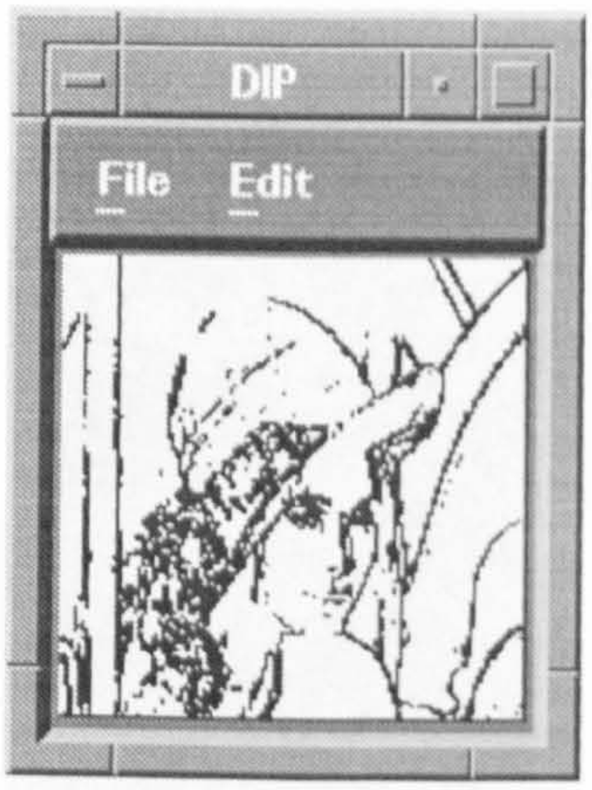


Mona: Prewitt edge detection (y - direction)

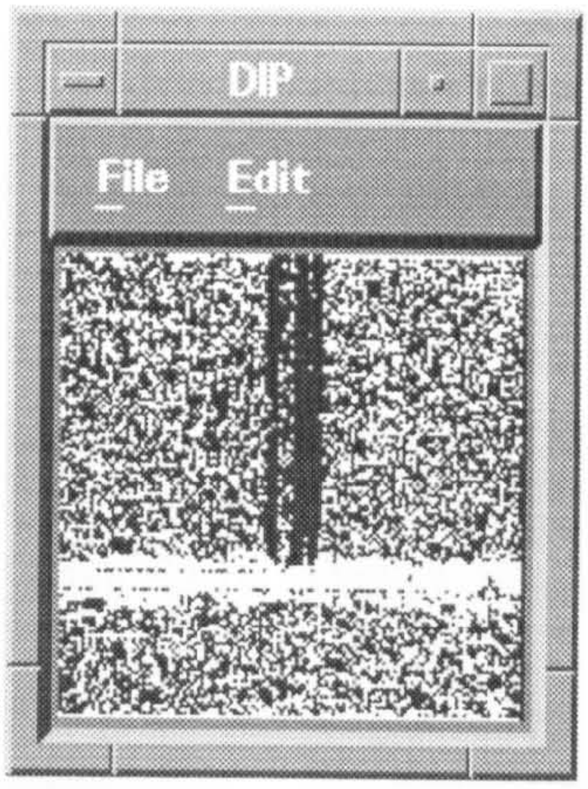


Mona: Roberts edge detection

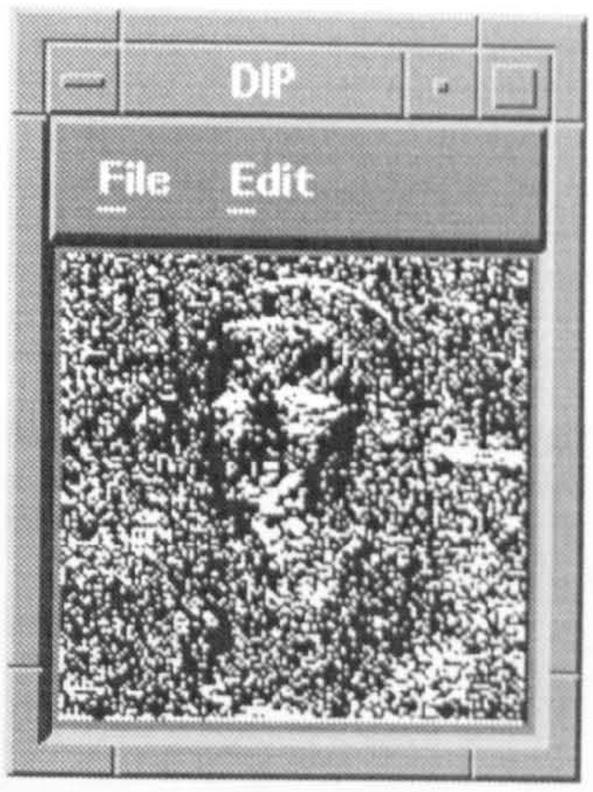
Lena: Orthogonal edge detection
(threshold=33)



Lena: Orthogonal edge detection
(threshold=55)

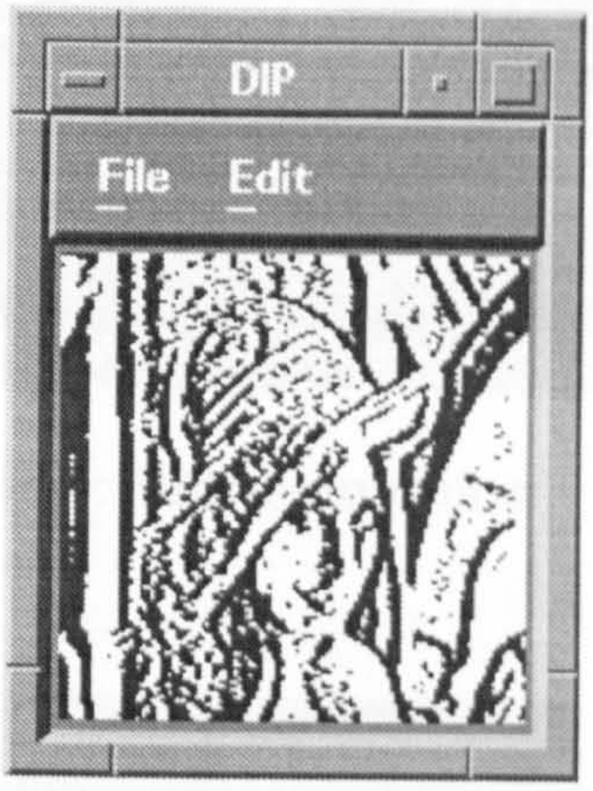


Tan inverse edge detection



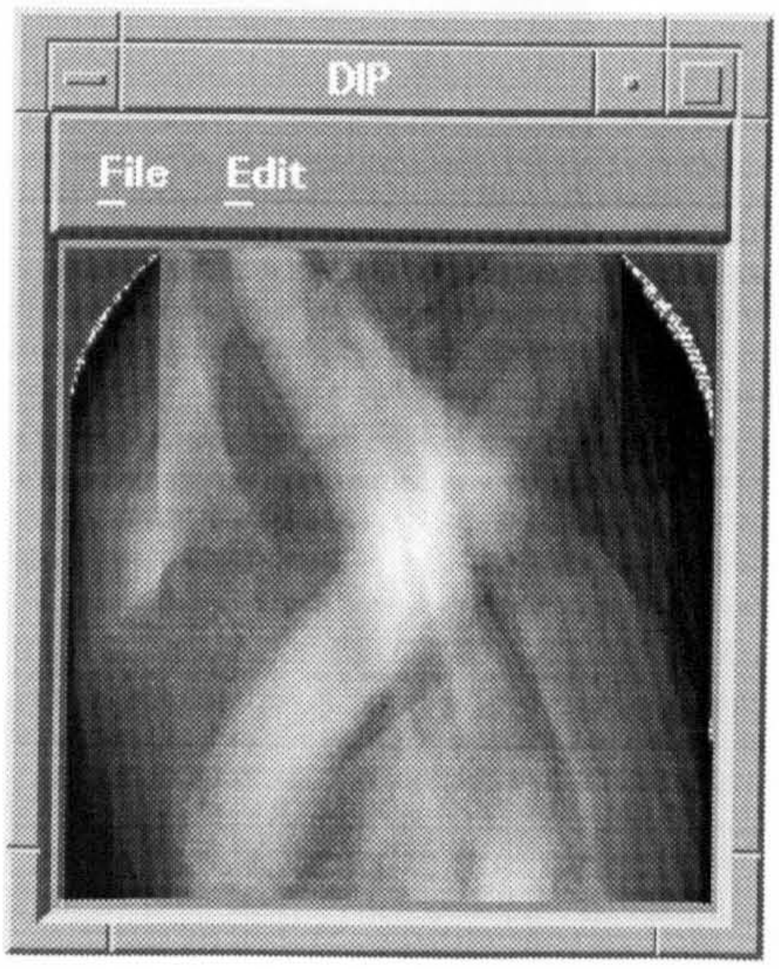
Mona: Tan inverse edge detection

Mona: ANN edge detection



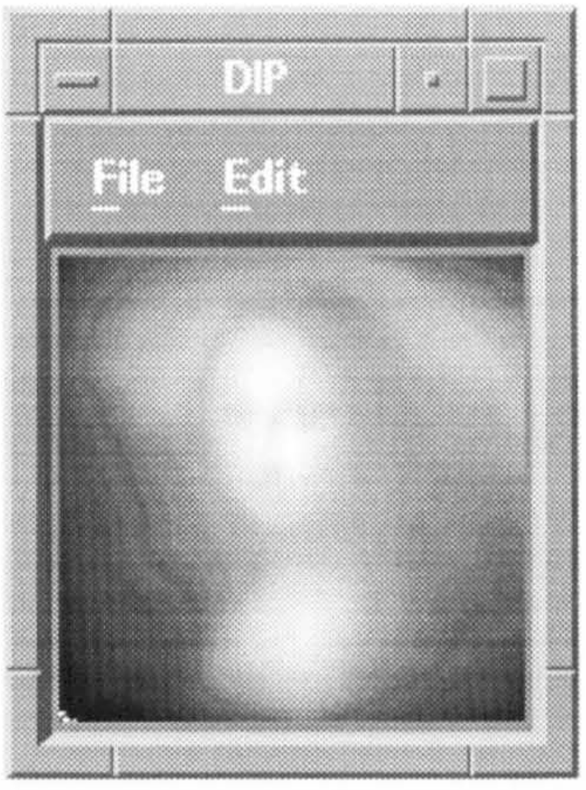
Lena: ANN edge detection

D.10. Results using Radon Menu option

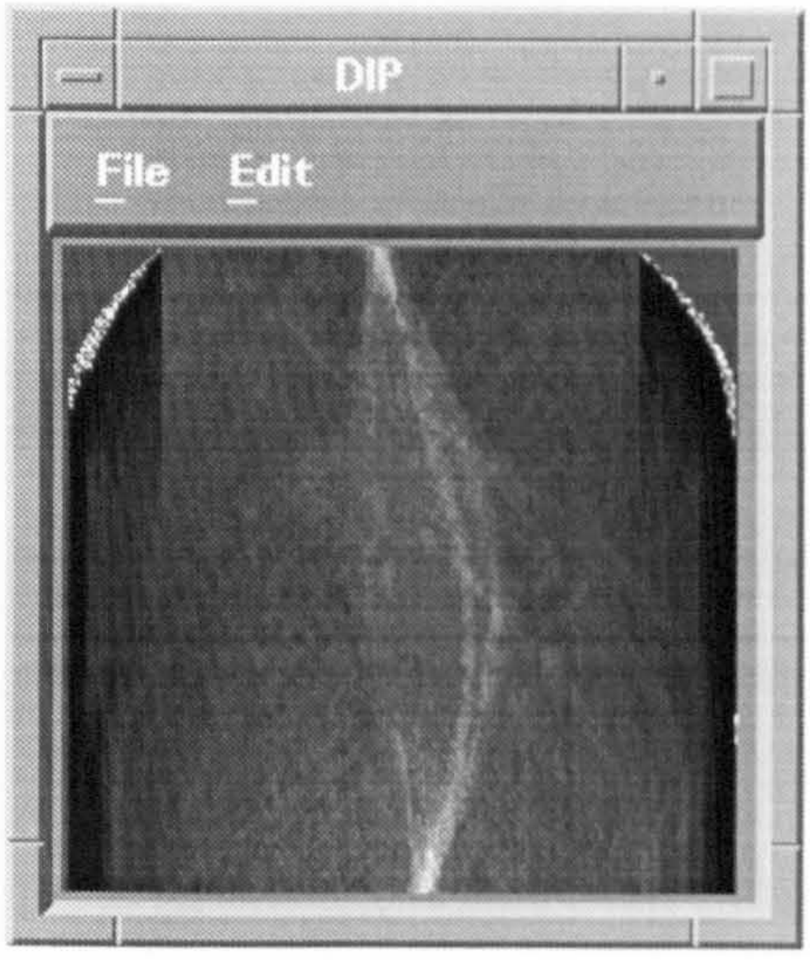


Mona: Back-projection/

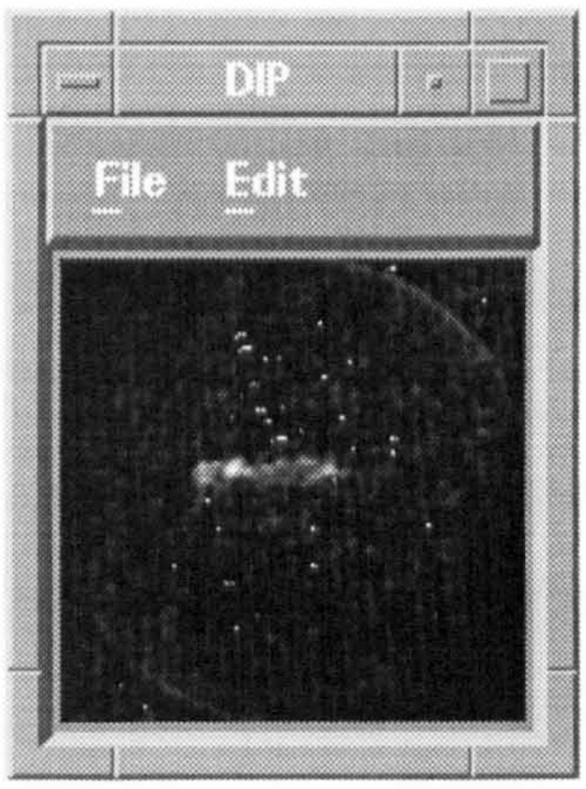
centre slice
Mona: Radon/Projection



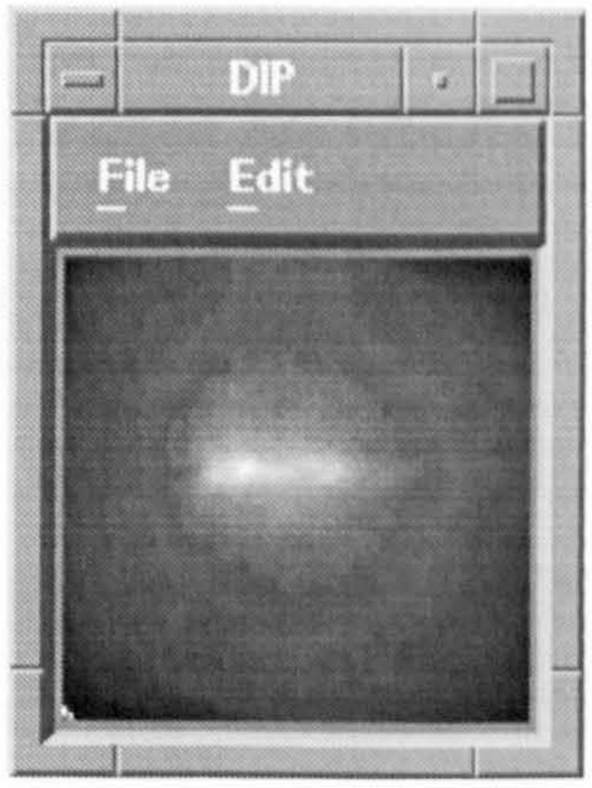
Mona: Back-projection/
deconvolution



Ship: Radon/projection



Ship: Radon/centre slice



Ship: Radon/deconvolution

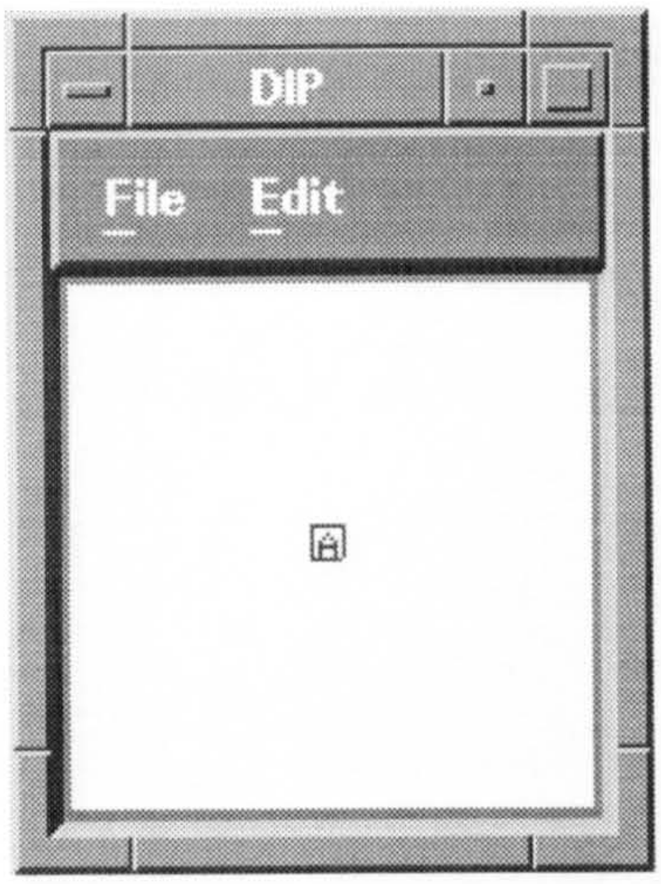
Appendix E

AUTOMATIC PATTERN RECOGNITION BASED FRACTAL COMPRESSION RESULTS

This appendix shows all the results of implementing the Automatic Pattern Recognition technique based on fractal image compression algorithm on black and white images and grey scale images. These results have been displayed in 2D space view and 3D space view.

In this appendix: **Page No**

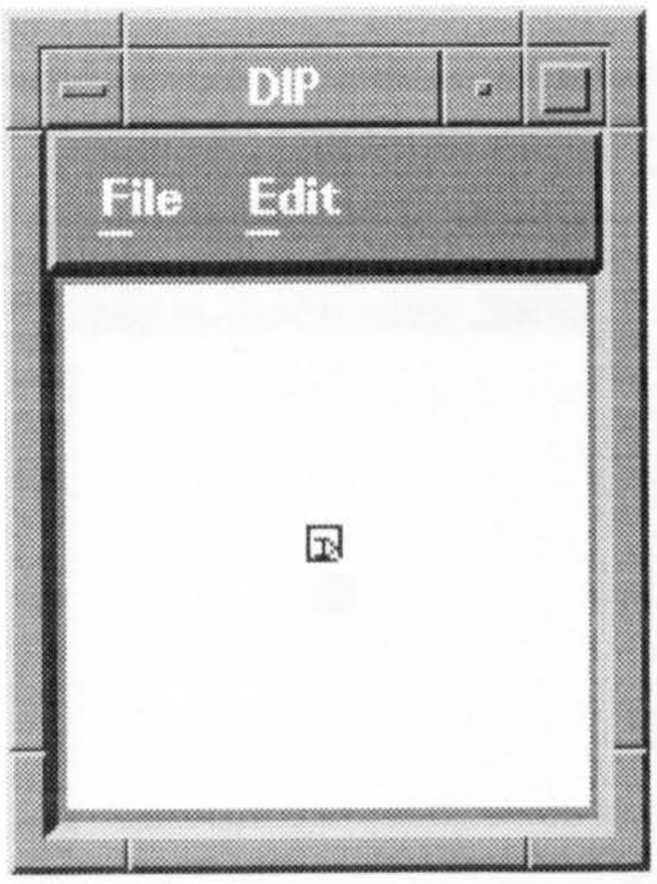
E.1: Results of implementing the algorithm on Black and White images:



222 222 222 0 222 222 222 222
222 222 0 222 0 222 222 222
222 0 222 222 222 0 222 222
222 0 0 0 0 0 222 222
222 0 222 222 222 0 222 222
222 0 222 222 222 0 222 222
222 222 222 222 222 222 222 222
222 222 222 222 222 222 222 222

0 0 2 9
0 0 2 37
0 0 1 65
0 0 1 65

Fig 7.5 A Block of image 1 **Data Block** **A Compressed Block**

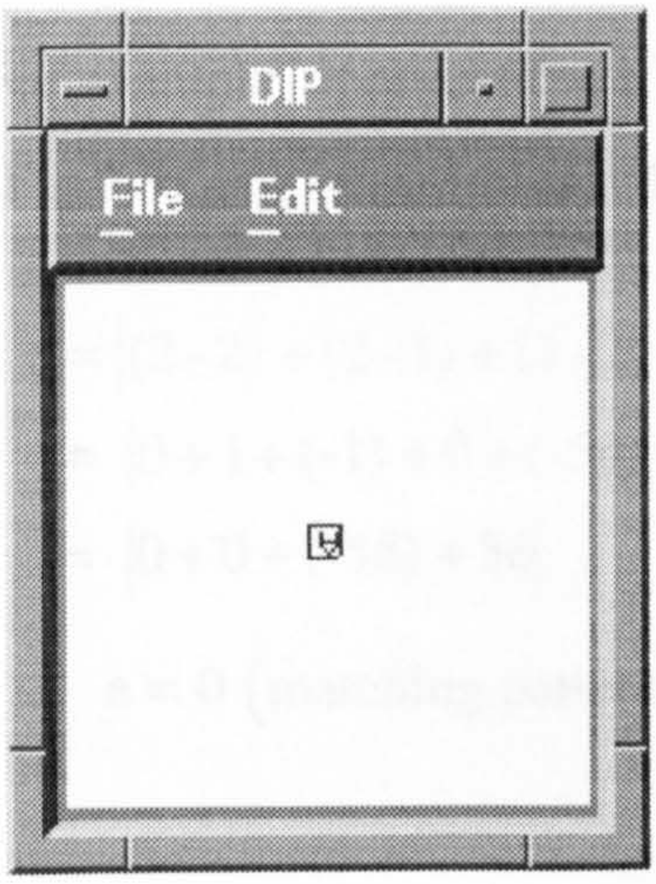


222 222 222 222 222 222 222 222
222 222 0 0 0 0 222 222
222 222 222 222 0 222 0 222
222 222 222 222 0 222 222 0
222 222 222 222 0 222 0 222
222 222 0 0 0 0 222 222
222 222 222 222 222 222 222 222
222 222 222 222 222 222 222 222

0 0 2 65
0 0 1 9
0 0 2 65
0 0 1 37

Fig 7.6 Rotated Block (90°) **Data Block**

A Compressed Block

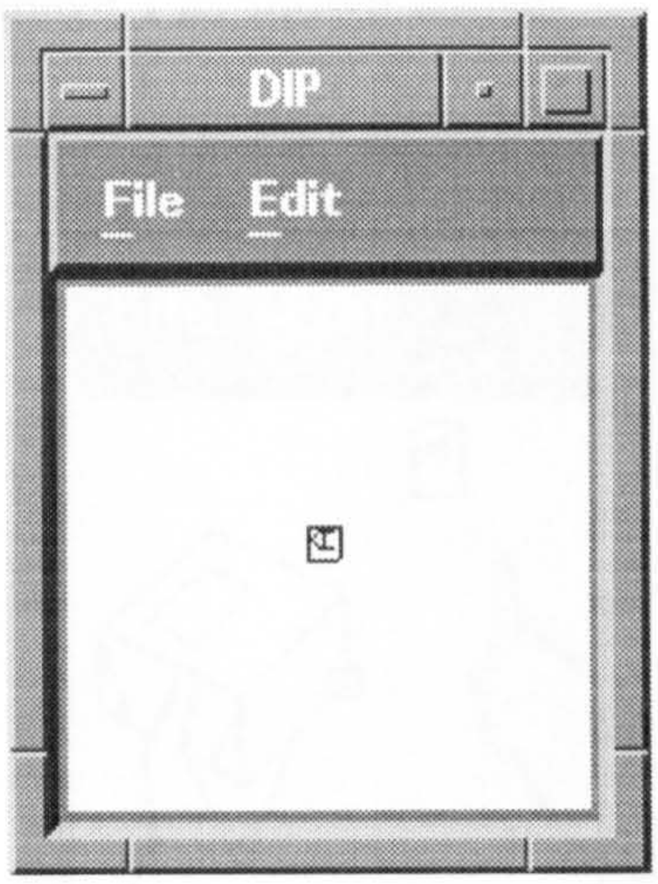


222 222 222 222 222 222 222 222
222 222 222 222 222 222 222 222
222 222 0 222 222 222 0 222
222 222 0 222 222 222 0 222
222 222 0 0 0 0 0 222
222 222 0 222 222 222 0 222
222 222 222 0 222 0 222 222
222 222 222 222 0 222 222 222

0 0 1 65
0 0 1 65
0 0 2 37
0 0 2 9

Fig 7.7 Rotated Block (180°) **Data**

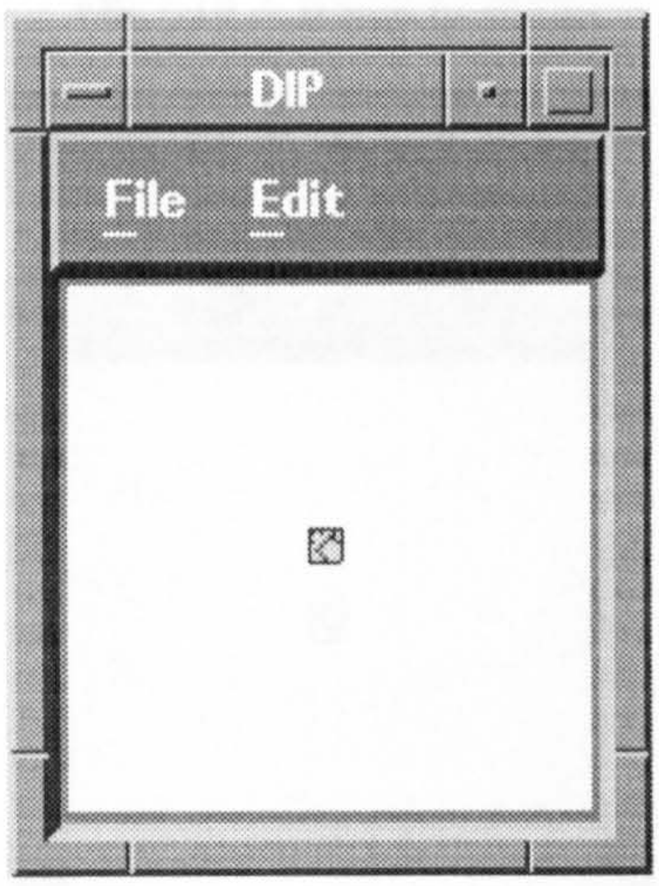
Block **A Compressed Block**



222 222 222 222 222 222 222 222
222 222 222 222 222 222 222 222
222 222 0 0 0 0 222 222 0 0 1 37
222 0 222 0 222 222 222 222 0 0 2 65
0 222 222 0 222 222 222 222 0 0 1 9
222 0 222 0 222 222 222 222 0 0 2 65
222 222 0 0 0 0 222 222
222 222 222 222 222 222 222 222

Fig 7.8 Rotated Block (270°) **Data**

Block **A Compressed Block**



0 222 0 222 222 0 222 222
222 222 222 222 0 0 222 222
0 222 222 0 222 222 0 222 0 0 2 23
222 222 0 222 222 222 222 0 0 0 1 23
222 0 222 222 222 222 222 222 0 0 1 51
222 222 0 222 222 222 222 222 0 0 2 79
222 222 222 0 222 222 222 222
222 222 222 222 0 222 222 222

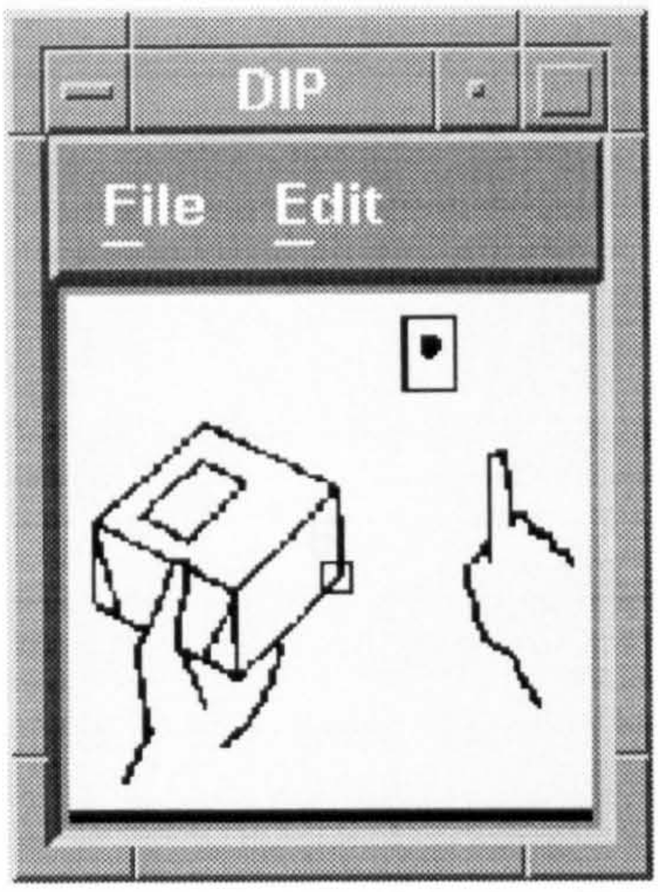
Fig 7.9 Rotated Block (45°) **Data Block** **A Compressed Block**

For example, applying the L^1 matching equation between the original image 1 (**Fig 7.5**) and rotated block (**Fig 7.6**), we get

$$e = \left|(2-2)+(2-1)+(1-2)+(1-1)+(9-65)+(37-9)+(65-65)+(65-37)\right|$$
$$= \left|0+1+(-1)+0+(-56)+28+0+28\right|$$
$$= \left|0+0+(-56)+56\right|$$

$\therefore \quad e = 0$ (matching pattern)

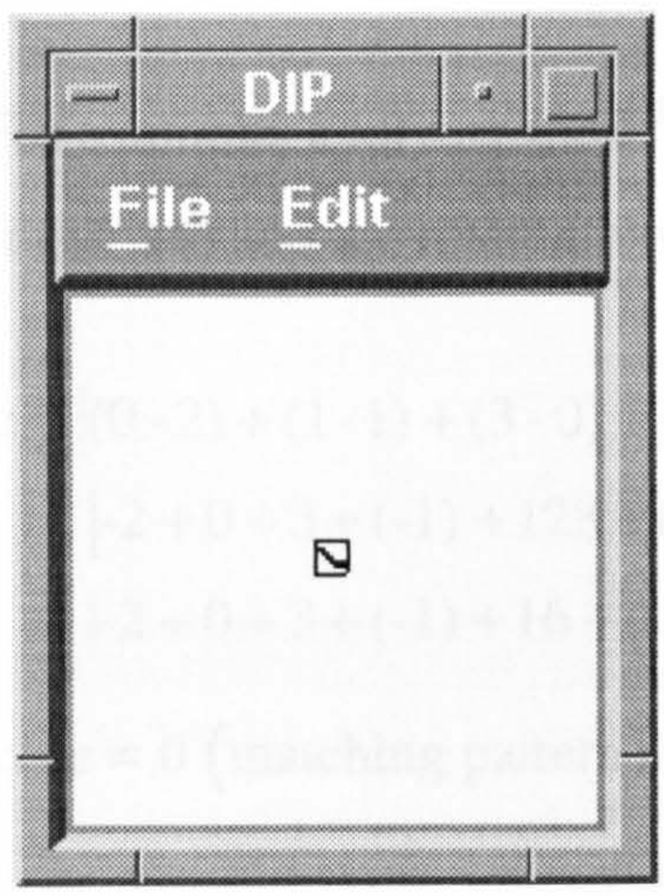
We get the same result, with the rest of all the different rotation Blocks of image 1 as shown above.



255 255 255 255 0 0 255 255
222 222 222 222 0 0 222 222
255 255 255 255 0 0 255 255
222 222 222 222 0 0 222 222
255 255 255 0 0 255 255 255
255 255 0 0 255 255 255 255
255 0 0 255 255 255 255 255
255 0 255 255 255 255 255 255

0 0 0 109
0 0 1 -19
0 0 3 13
0 0 2 93

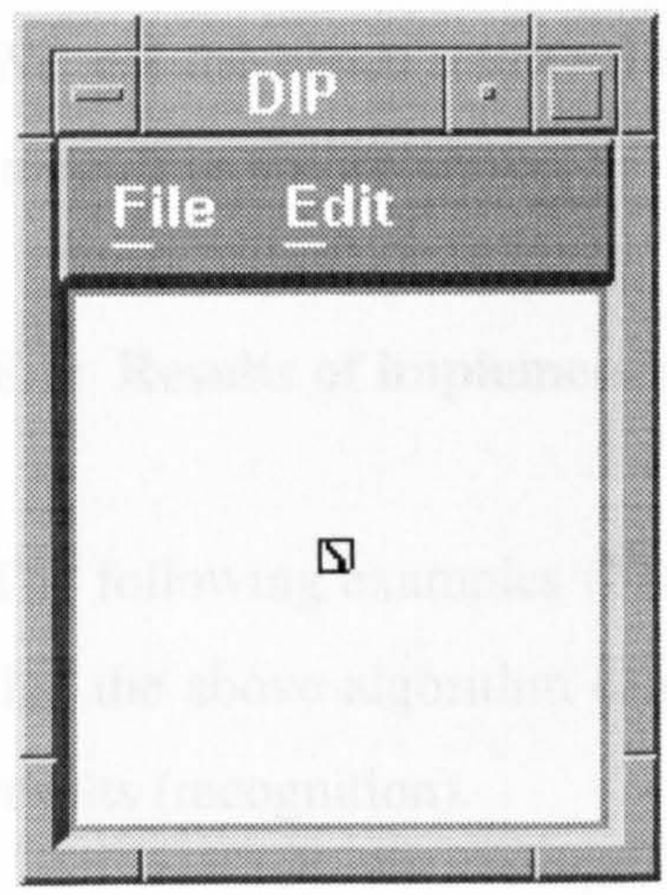
Fig 7.10 A Block of image 2 **Data Block** **A Compressed Block**



255 255 255 255 255 255 255 255
0 0 255 255 255 255 255 255
255 0 0 255 255 255 255 255
255 255 0 0 255 255 255 255
255 255 255 0 0 0 0 0
255 255 255 255 0 0 0 0
255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255

0 0 3 13
0 0 0 109
0 0 1 93
0 0 2 -19

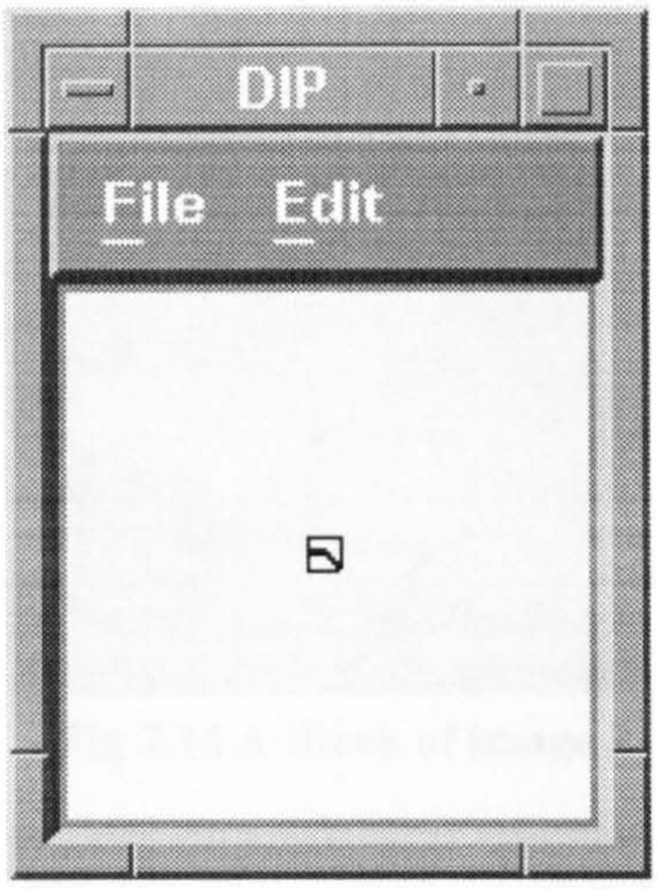
Fig 7.11 Rotated Block (90°) **Data Block** **A Compressed Block**



255 255 255 255 0 0 255 255
222 222 222 222 0 0 222 222
255 255 255 255 0 0 255 255
222 222 222 222 0 0 222 222
255 255 255 0 0 255 255 255
255 255 0 0 255 255 255 255
255 0 0 255 255 255 255 255
255 0 255 255 255 255 255 255

0 0 0 109
0 0 1 -19
0 0 3 13
0 0 2 93

Fig 7.12 Rotated Block (180°) **Data Block** **A Compressed Block**



255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255
0 0 0 0 255 255 255 255
0 0 0 0 0 255 255 255 0 0 0 109
255 255 255 255 0 0 255 255 0 0 1 -19
255 255 255 255 255 0 0 255 0 0 3 13
255 255 255 255 255 255 0 0 0 0 2 93
255 255 255 255 255 255 255 255

Fig 7.13 Rotated Block (270°) **Data Block** **A Compressed Block**

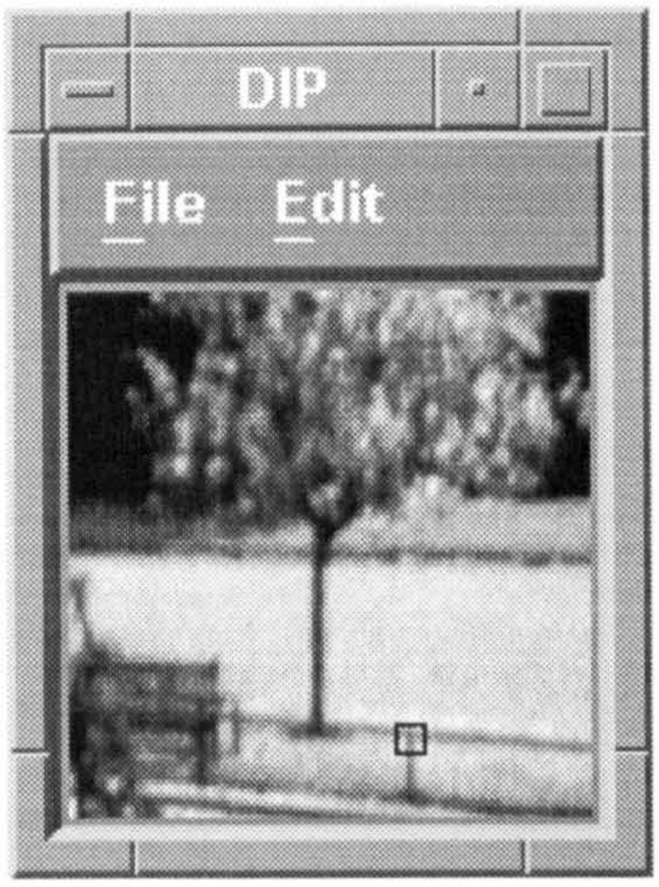
For example, applying the L^1 matching equation between the original image 2 (**Fig 7.10**) and the rotated block (**Fig 7.13**), we get

$$e = \left|(0-2)+(1-1)+(3-0)+(2-3)+(109-(-19))+(-19-93)+(13-109)+(93-13)\right|$$
$$= \left|-2+0+3+(-1)+128+(-112)+(-96)+80\right|$$
$$= \left|-2+0+3+(-1)+16+(-16)\right|$$
$$\therefore \quad e = 0 \ (\text{matching pattern})$$

We get the same result with the rest of all the different rotation Blocks of image 2 as shown above.

E.2: Results of implementing the algorithm on greyscale images:

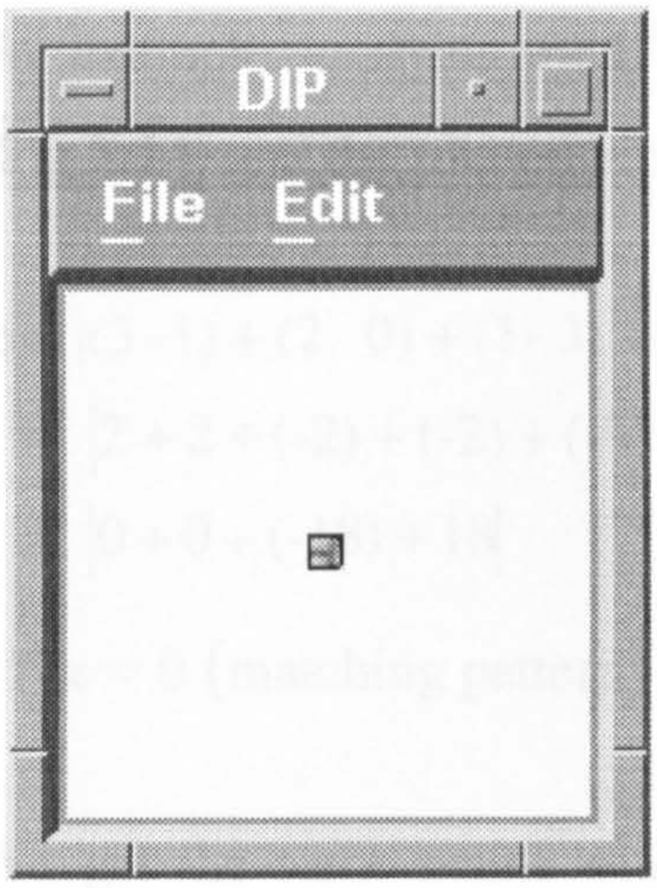
The following examples (**Fig 7.14**) to (**Fig 7.33**) of greyscales images, shows that the above algorithm can operate on these type of images and gives good results (recognition).



186 190 200 210 220 220 210 208
156 154 162 160 174 186 182 180
138 138 136 114 132 138 146 150
168 164 150 120 122 138 136 148
204 192 172 124 128 164 170 174
202 202 174 128 136 182 192 202
202 202 166 136 144 200 204 206
202 200 172 126 150 182 202 210

0 0 3 32
0 0 2 41
0 0 1 48
0 0 0 50

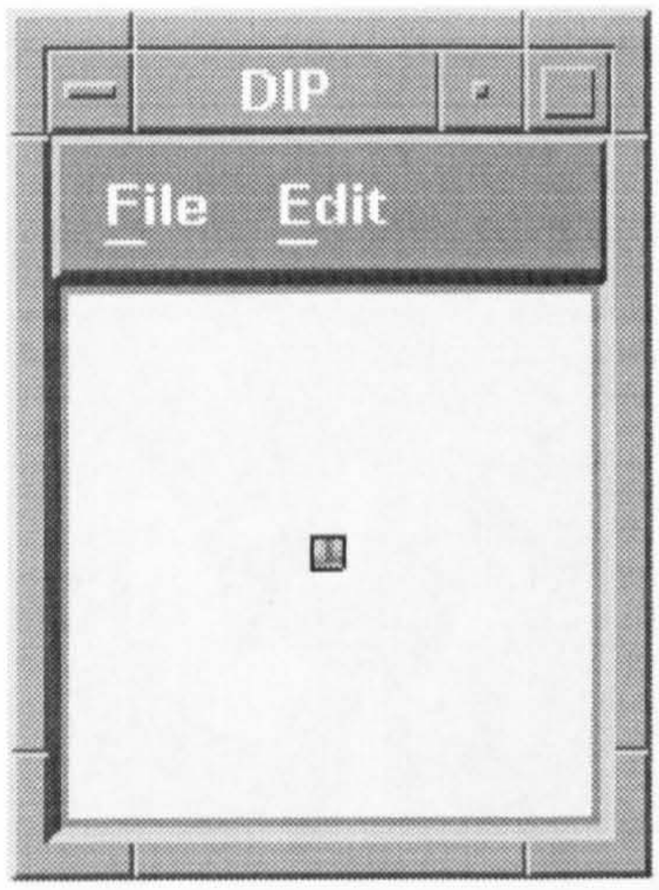
Fig 7.14 A Block of image 3 **Data Block** **A Compressed Block**



208 180 150 148 174 202 206 210
210 182 146 136 170 192 204 202
220 186 138 138 164 182 200 182
220 174 132 122 128 136 144 150
210 160 114 120 124 128 136 126
200 162 136 150 172 174 166 172
190 154 138 164 192 202 202 200
186 156 138 168 204 202 202 202

0 0 1 41
0 0 0 50
0 0 3 32
0 0 2 48

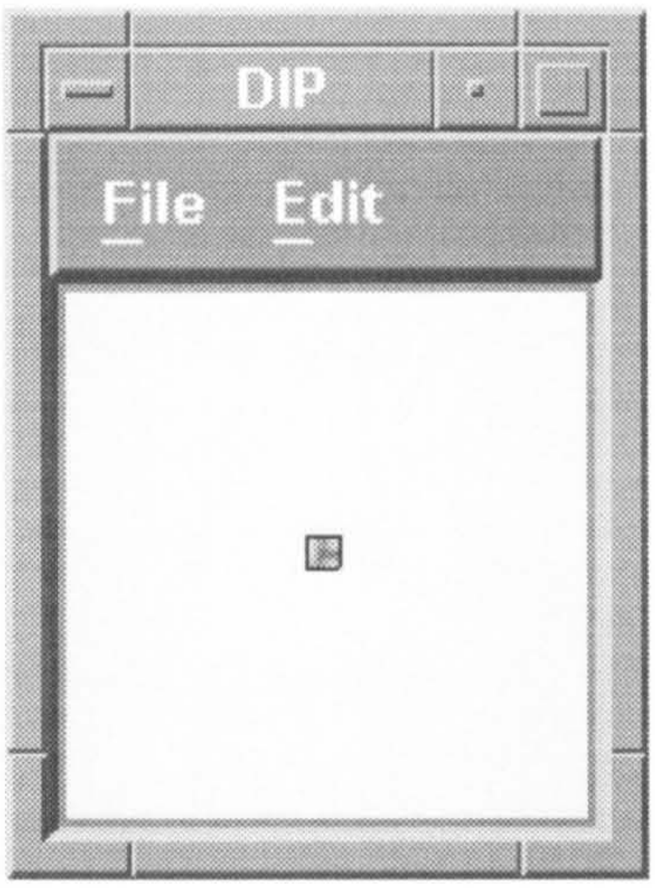
Fig 7.15 Rotated Block (90°) **Data Block** **A Compressed Block**



202 200 172 126 150 182 202 210
202 202 166 136 144 200 204 206
202 202 174 128 136 182 192 202
204 192 172 124 128 164 170 174
168 164 150 120 122 138 136 148
138 138 136 114 132 138 146 150
154 154 162 160 174 186 182 180
186 190 200 210 220 220 210 208

0 0 1 48
0 0 0 50
0 0 3 32
0 0 2 41

Fig 7.16 Rotated Block (180°) **Data Block** **A Compressed Block**



202 202 202 204 168 138 156 186
200 202 202 192 164 138 154 190
172 166 174 172 150 136 162 200
126 136 128 124 120 114 160 210
150 144 136 128 122 132 174 220
182 200 182 164 138 138 186 220
202 204 192 170 136 146 182 210
210 206 202 174 148 150 180 208

0 0 2 48
0 0 3 -32
0 0 0 50
0 0 1 41

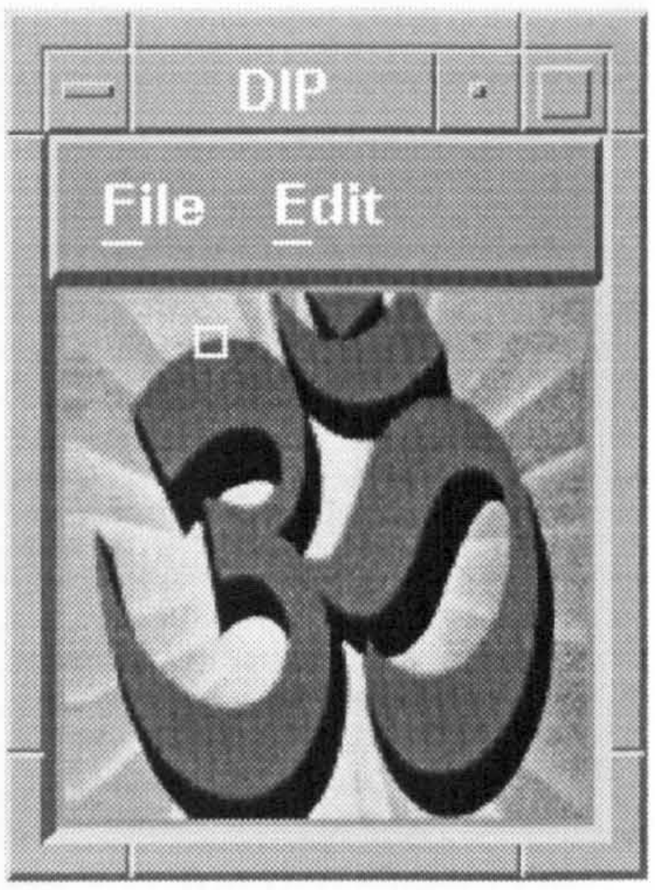
Fig 7.17 Rotated Block (270°) **Data Block** **A Compressed Block**

For example, applying the L^1 matching equation between the original image 3 (**Fig 7.14**) and the rotated block (**Fig 7.16**), we get

$$e = \left|(3-1)+(2-0)+(1-3)+(0-2)+(32-41)+(41-50)+(48-32)+(50-48)\right|$$
$$= \left|2+2+(-2)+(-2)+(-9)+(-9)+16+2\right|$$
$$= \left|0+0+(-18)+18\right|$$

$$\therefore \quad e = 0 \left(\text{matching pattern}\right)$$

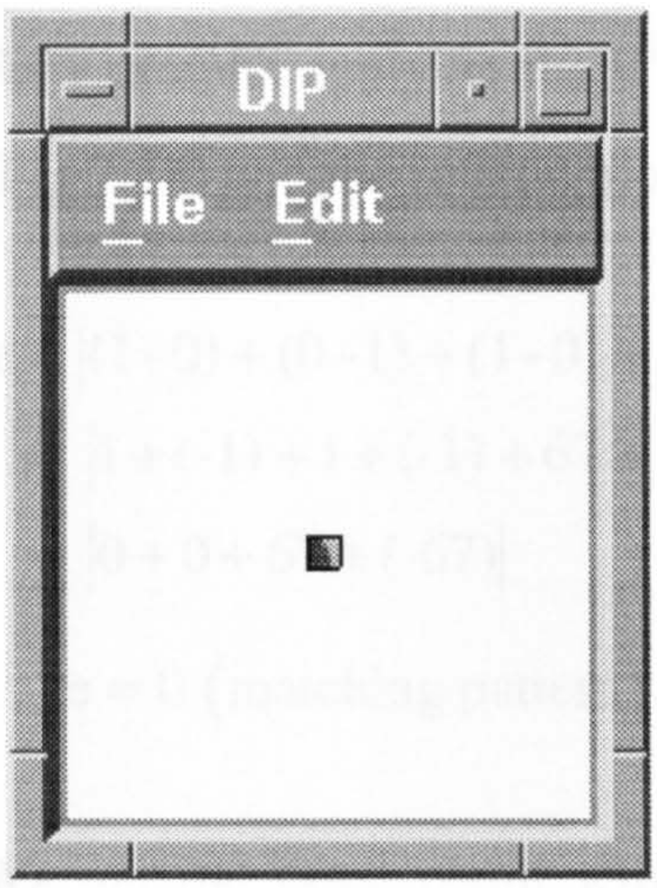
We get the same result with the rest of all the different rotation Blocks of image 3 as shown above.



204 198 198 204 198 198 198 210
198 198 192 188 198 198 198 204
198 192 192 182 182 176 176 192
192 188 176 144 120 104 98 98
176 138 104 104 88 98 110 94
110 94 94 104 98 98 94 94
94 94 98 94 82 94 98 94
82 94 94 98 98 98 98 94

0 0 1 67
0 0 0 86
0 0 1 -9
0 0 0 0

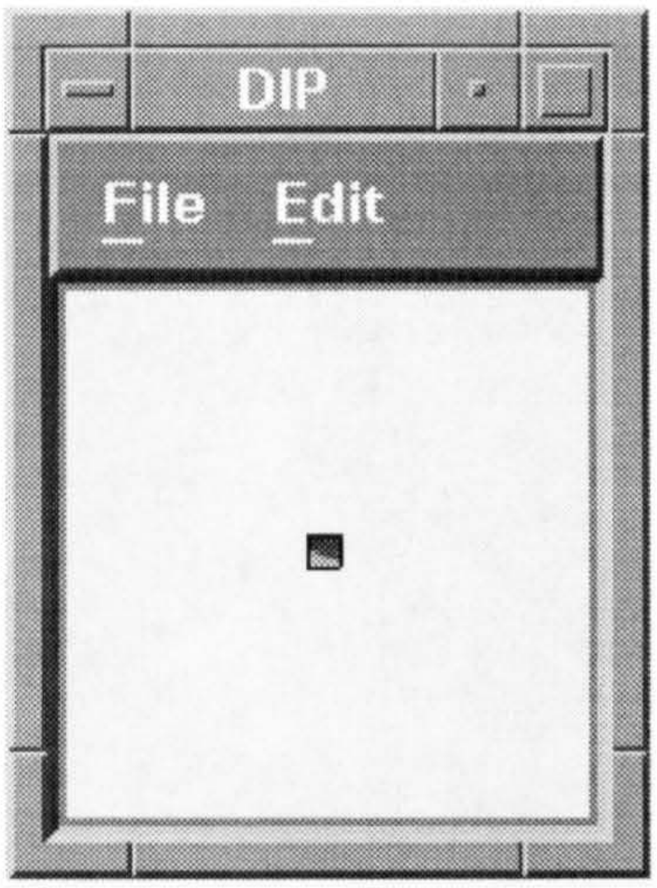
Fig 7.18 A Block of image 4 **Data Block** **A Compressed Block**



210 204 192 98 94 94 94 94
198 198 176 98 110 94 98 98
198 198 176 104 98 98 94 98
198 198 182 120 88 98 82 98
204 188 182 144 104 104 94 98
198 192 192 176 104 94 98 94
198 198 192 188 138 94 94 94
204 198 198 192 176 110 94 82

0 0 1 67
0 0 1 -9
0 0 0 86
0 0 0 0

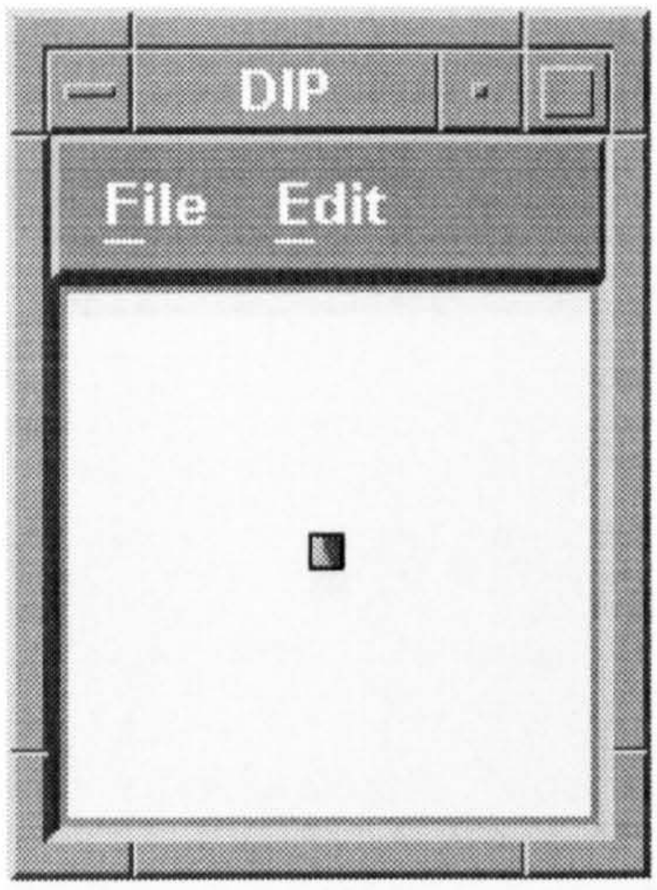
Fig 7.19 Rotated Block (90°) **Data Block** **A Compressed Block**



82 94 94 98 98 98 98 94
94 94 98 94 82 94 98 94
110 94 94 104 98 98 94 94
176 138 104 104 88 98 110 94
192 188 176 144 120 104 98 98
198 192 192 182 182 176 176 192
198 198 192 188 198 198 198 204
204 198 198 204 198 198 198 210

0 0 0 0
0 0 1 -9
0 0 0 86
0 0 1 67

Fig 7.20 Rotated Block (180°) **Data Block** **A Compressed Block**



82 94 110 176 192 198 198 204
94 94 94 138 188 192 198 198
94 98 94 104 176 192 192 198
98 94 104 104 144 182 188 204
98 82 98 88 120 182 198 198
98 94 98 98 104 176 198 198
98 98 94 110 98 176 198 198
94 94 94 94 98 192 204 210

0 0 0 0
0 0 0 86
0 0 1 -9
0 0 1 67

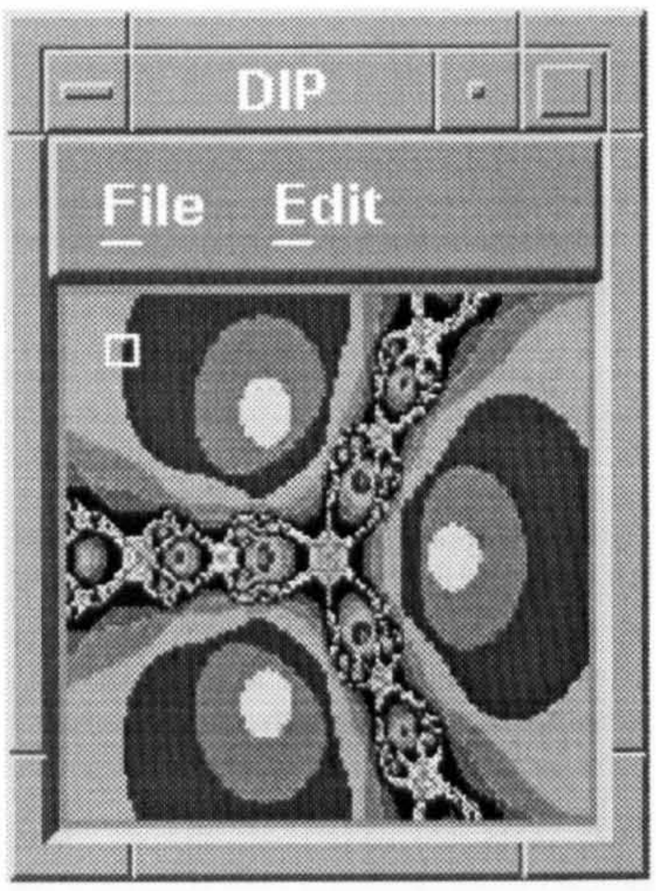
Fig 7.21 Rotated Block (270°) **Data Block** **A Compressed Block**

For example, applying the L^1 matching equation between the original image 4 (**Fig 7.18**) and the rotated block (**Fig 7.20**), we get

$$e = \left|(1-0)+(0-1)+(1-0)+(0-1)+(67-0)+(86-(-9))+((-9)-86)+(0-67)\right|$$
$$= \left|1+(-1)+1+(-1)+67+95+(-95)+(-67)\right|$$
$$= \left|0+0+67+(-67)\right|$$

$$\therefore \quad e = 0 \left(\text{matching pattern}\right)$$

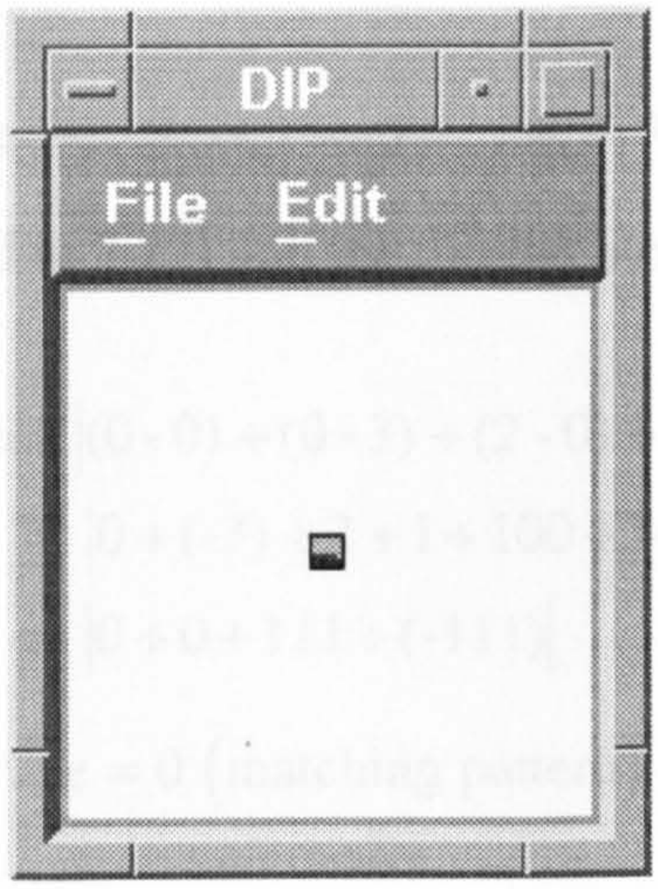
We get the same result with the rest of all the different rotation Blocks of image 4 as shown above.



177 177 177 177 182 71 60 52
177 177 177 177 190 66 57 60
177 177 177 177 54 61 62 67
177 177 177 177 63 61 63 65
177 184 172 169 60 63 63 64
176 177 169 168 61 64 64 64
174 169 177 173 61 64 64 64
173 179 177 182 62 64 64 64

0 0 0 86
0 0 0 -14
0 0 2 83
0 0 3 -28

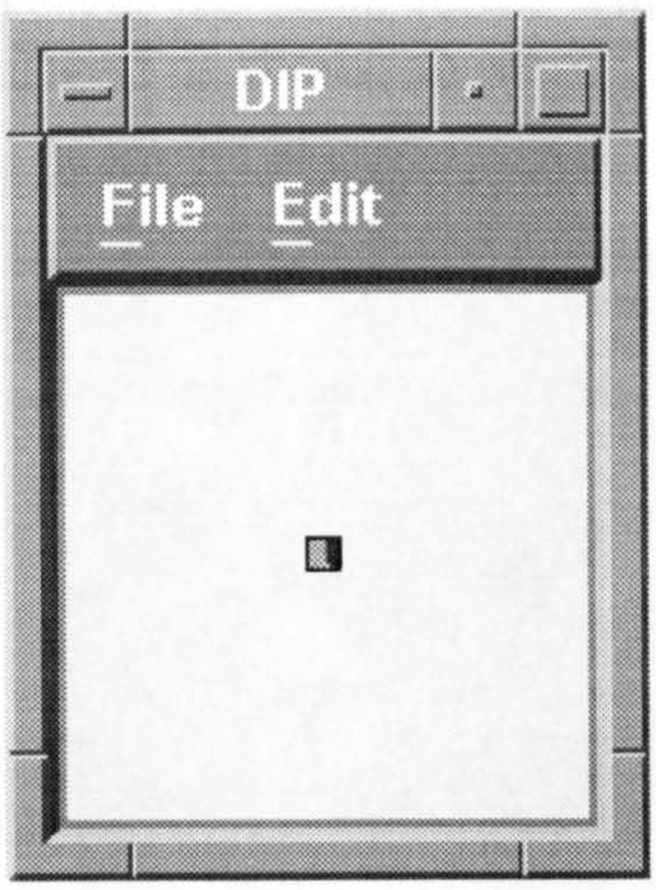
Fig 7.22 A Block of image 5 **Data Block** **A Compressed Block**



52 60 67 65 64 64 64 64
60 57 62 63 63 64 64 64
71 66 61 61 63 64 64 64
182 190 54 63 60 61 61 62
177 177 177 177 169 168 173 182
177 177 177 177 172 169 177 177
177 177 177 177 184 177 169 179
177 177 177 177 177 176 174 173

0 0 0 -14
0 0 3 -28
0 0 0 86
0 0 2 83

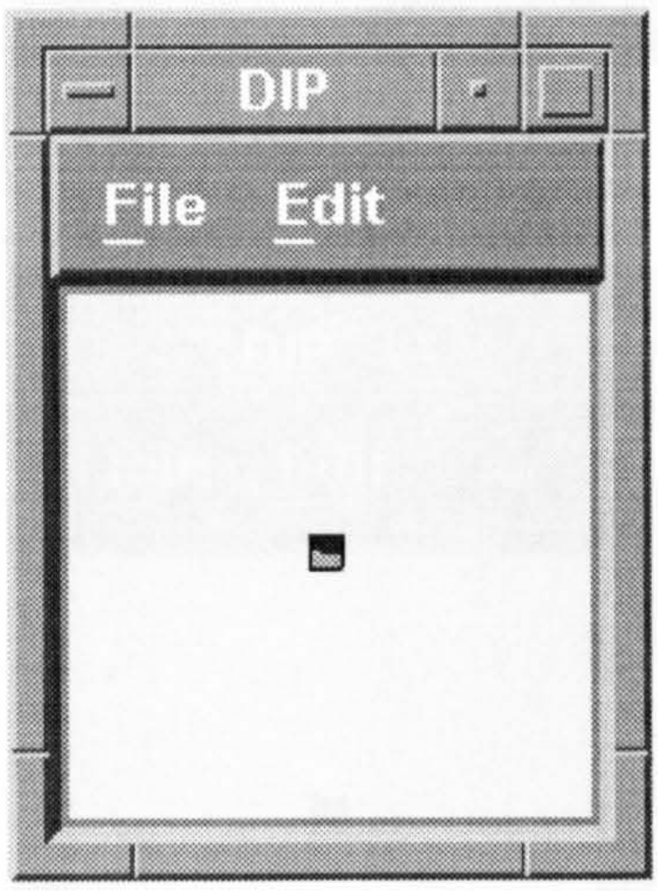
Fig 7.23 Rotated Block (90°) **Data Block** **A Compressed Block**



173 179 177 182 62 64 64 64
174 169 177 173 61 64 64 64
176 177 169 168 61 64 64 64
177 184 172 169 60 63 63 64
177 177 177 177 63 61 63 65
177 177 177 177 54 61 62 67
177 177 177 177 190 66 57 60
177 177 177 177 182 71 60 52

0 0 2 83
0 0 3 -28
0 0 0 86
0 0 0 -14

Fig 7.24 Rotated Block (180°) **Data Block** **A Compressed Block**



173 174 176 177 177 177 177 177
179 169 177 184 177 177 177 177
177 177 169 172 177 177 177 177
182 173 168 169 177 177 177 177
62 61 61 60 63 54 190 182
64 64 64 63 61 61 66 71
64 64 64 63 63 62 57 60
64 64 64 64 65 67 60 52

0 0 2 83
0 0 0 86
0 0 3 -28
0 0 0 -14

Fig 7.25 Rotated Block (270°) **Data Block** **A Compressed Block**

For example, applying the L^1 matching equation between the original image 5 (**Fig 7.22**) and the rotated block (**Fig 7.23**), we get

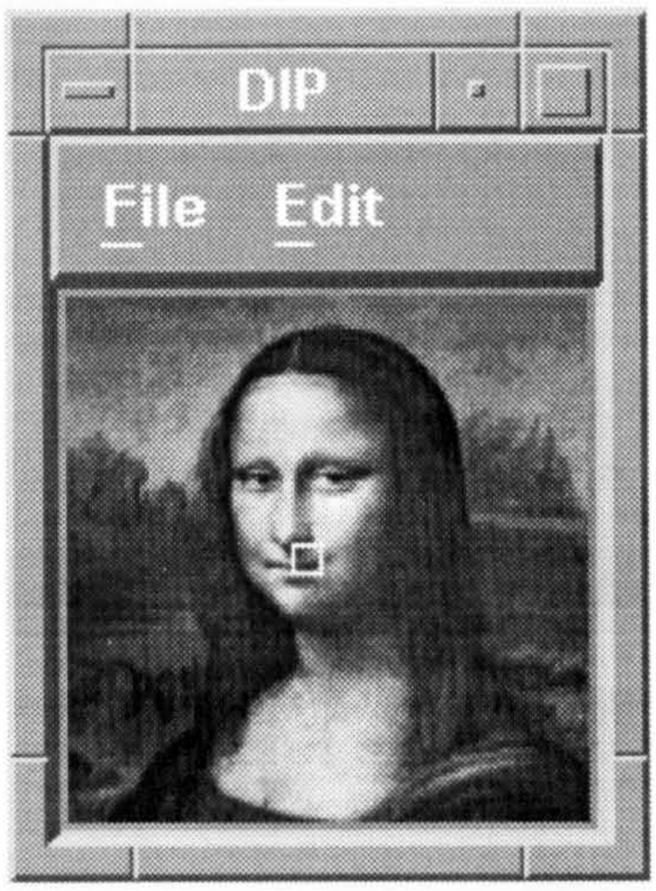
$$e = \left|(0-0)+(0-3)+(2-0)+(3-2)+(86-(-14))+((-14)-(-28))+(83-86)+((-28)-83)\right|$$

$$= \left|0+(-3)+2+1+100+14+(-3)+(-111)\right|$$

$$= \left|0+0+111+(-111)\right|$$

$$\therefore \quad e = 0 \ \text{(matching pattern)}$$

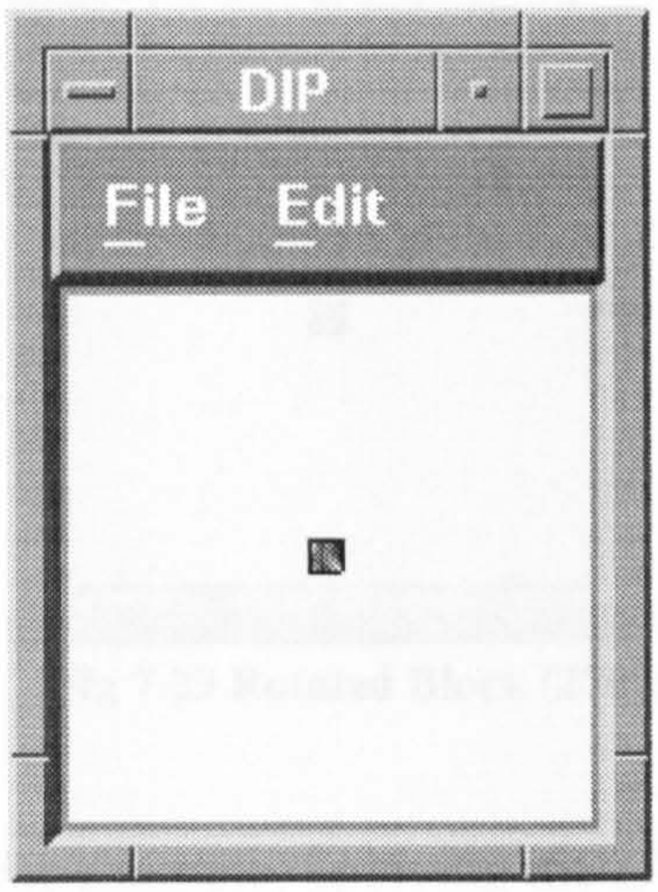
We get the same result with the rest of all the different rotation Blocks of image 5 as shown above.



66 55 57 63 65 133 215 231
60 60 74 87 113 156 209 209
87 91 105 128 145 174 179 170
135 127 132 137 152 160 149 134
148 133 134 135 135 118 110 95
106 88 87 87 87 72 54 83
128 128 113 112 105 122 151 178
216 193 157 142 133 144 167 168

0 0 1 -3
0 0 0 68
0 0 3 37
0 0 2 25

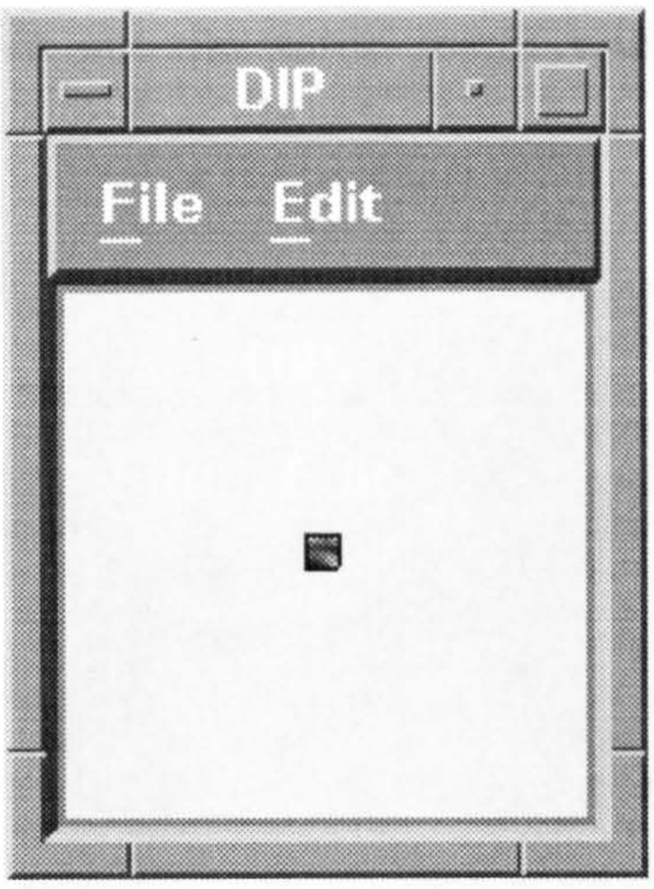
Fig 7.26 A Block of image 6 **Data Block** **A Compressed Block**



231 209 170 134 95 83 178 168
215 209 179 149 110 54 151 167
133 156 174 160 118 72 122 144
65 113 145 152 135 78 105 133
63 87 128 137 135 87 112 142
57 74 105 132 134 87 113 157
55 60 91 127 133 88 128 193
66 60 87 135 148 106 128 216

0 0 0 68
0 0 1 25
0 0 2 -3
0 0 3 37

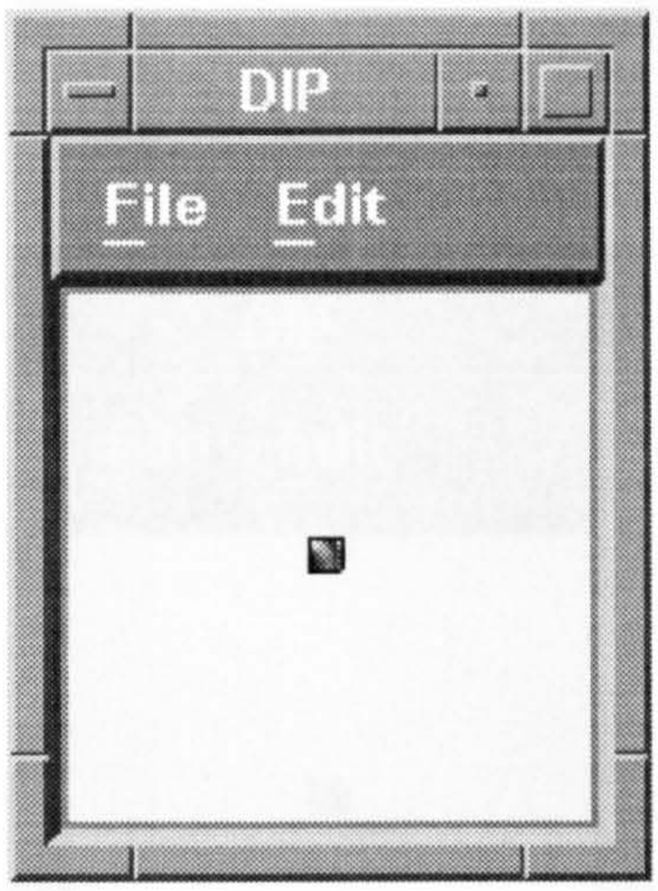
Fig 7.27 Rotated Block (90°) **Data Block** **A Compressed Block**



216 193 157 142 133 144 167 168
128 128 113 112 105 122 151 178
106 88 87 87 87 72 54 83
148 133 134 135 135 118 110 95
135 127 132 137 152 160 149 134
87 91 105 128 145 174 179 170
60 60 74 87 113 156 209 209
66 55 57 63 65 133 215 231

0 0 3 37
0 0 2 25
0 0 1 -3
0 0 0 68

Fig 7.28 Rotated Block (180°) **Data Block** **A Compressed Block**



216 128 106 148 135 87 60 66
193 128 88 133 127 91 60 55
157 113 87 134 132 105 74 57
142 112 87 135 137 128 87 63
133 105 78 135 152 145 113 65
144 122 72 118 160 174 156 133
167 151 54 110 149 179 209 215
168 178 83 95 134 170 209 231

0 0 3 37
0 0 2 -3
0 0 1 25
0 0 0 68

Fig 7.29 Rotated Block (270°) **Data Block** **A Compressed Block**

For example, applying the L^1 matching equation between the original image 6 (**Fig 7.28**) and the rotated block (**Fig 7.26**), we get

$$e = \left|(1-3)+(0-2)+(3-1)+(2-0)+((-3)-37)+(68-25)+(37-(-3))+(25-68)\right|$$

$$= \left|(-2)+(-2)+2+2+(-40)+43+40+(-43)\right|$$

$$= \left|0+0+83+(-83)\right|$$

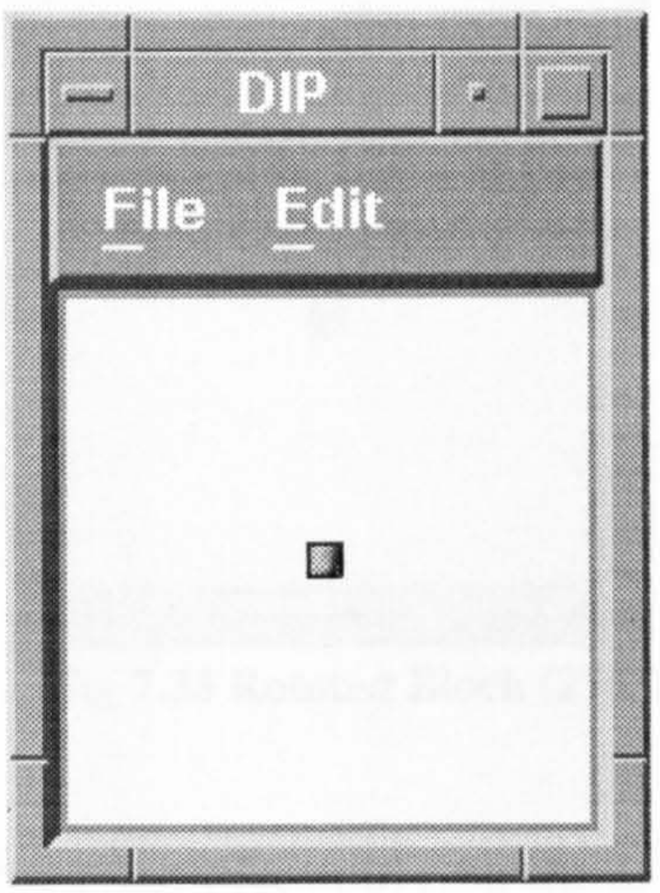
$$\therefore \quad e = 0 \ \text{(matching pattern)}$$

We get the same result with the rest of all the different rotation Blocks of image 6 as shown above.

85 114 122 130 158 158 92 87
104 68 56 51 70 85 106 131
153 117 101 73 69 67 56 79
172 147 141 131 137 122 72 62
193 174 163 166 175 174 137 118
205 202 196 186 188 184 165 153
196 199 188 188 185 177 171 140
170 172 172 178 189 187 174 139

0 0 0 7
0 0 3 -7
0 0 2 81
0 0 0 63

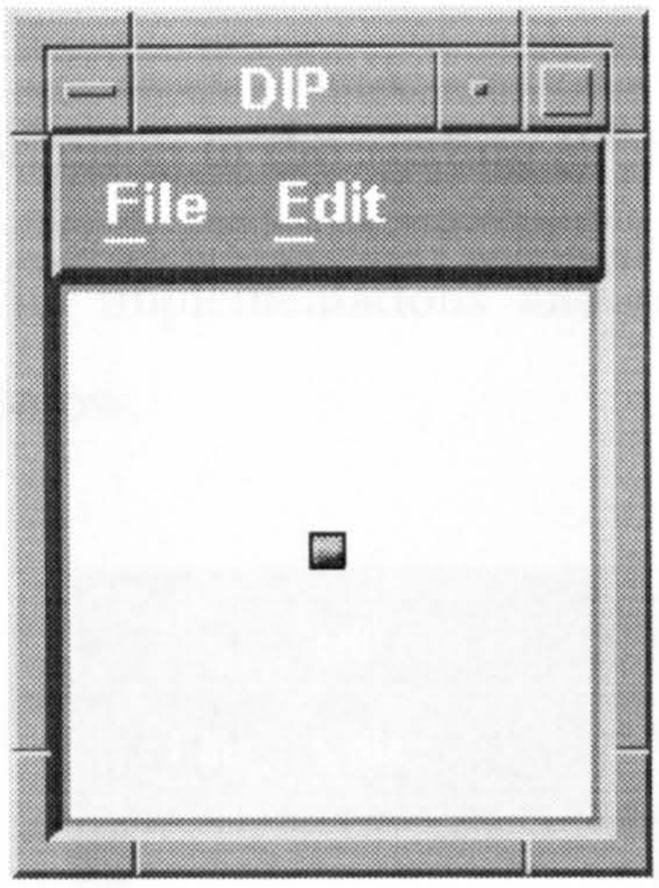
Fig 7.30 A Block of image 7 **Data Block** **A Compressed Block**



87 131 79 62 118 153 140 139
92 106 56 72 137 165 171 174
158 85 67 122 174 184 177 187
158 70 69 137 175 188 185 189
130 51 73 131 166 186 188 178
122 56 101 141 163 196 188 172
114 68 117 147 174 202 199 172
85 104 153 172 193 205 196 170

0 0 3 -7
0 0 0 63
0 0 0 7
0 0 2 81

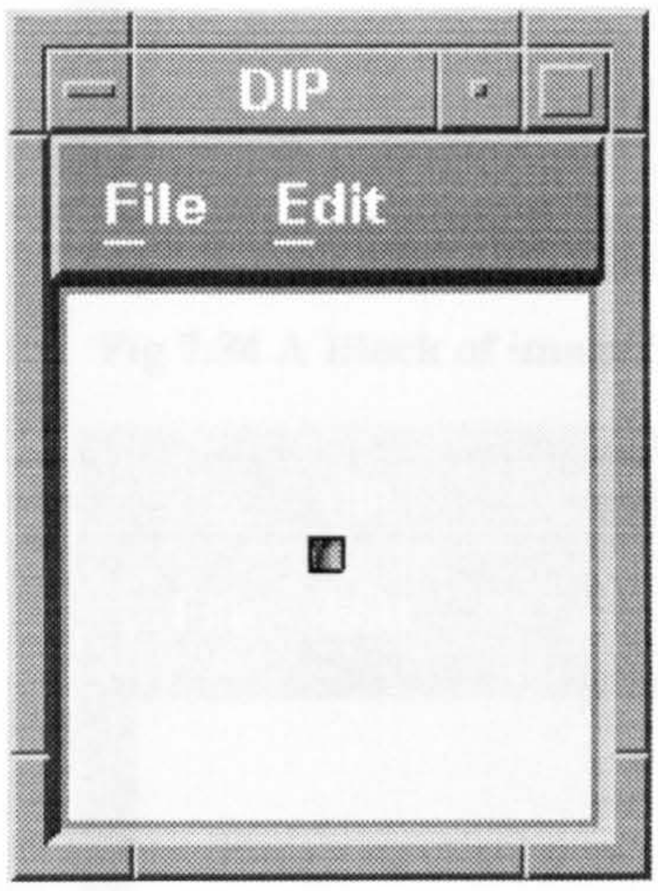
Fig 7.31 Rotated Block (90°) **Data Block** **A Compressed Block**



170 172 172 178 189 187 174 139
196 199 188 188 185 177 171 140
205 202 196 186 188 184 165 153
193 174 163 166 175 174 137 118
172 147 141 131 137 122 72 62
153 117 101 73 69 67 56 79
104 68 56 51 70 85 106 131
85 114 122 130 158 158 92 87

0 0 2 81
0 0 0 63
0 0 0 7
0 0 3 -7

Fig 7.32 Rotated Block (180°) **Data Block** **A Compressed Block**



170 196 205 193 172 153 104 85
172 199 202 174 147 117 68 114
172 188 196 163 141 101 56 122
178 188 186 166 131 73 51 130
189 185 188 175 137 69 70 158
187 177 184 174 122 67 85 158
174 171 165 137 72 56 106 92
139 140 153 118 62 79 131 87

0 0 2 81
0 0 0 7
0 0 0 63
0 0 3 -7

Fig 7.33 Rotated Block (270°) **Data Block** **A Compressed Block**

For example, applying the L^1 matching equation between the original image 7 (**Fig 7.30**) and the rotated block (**Fig 7.33**), we get

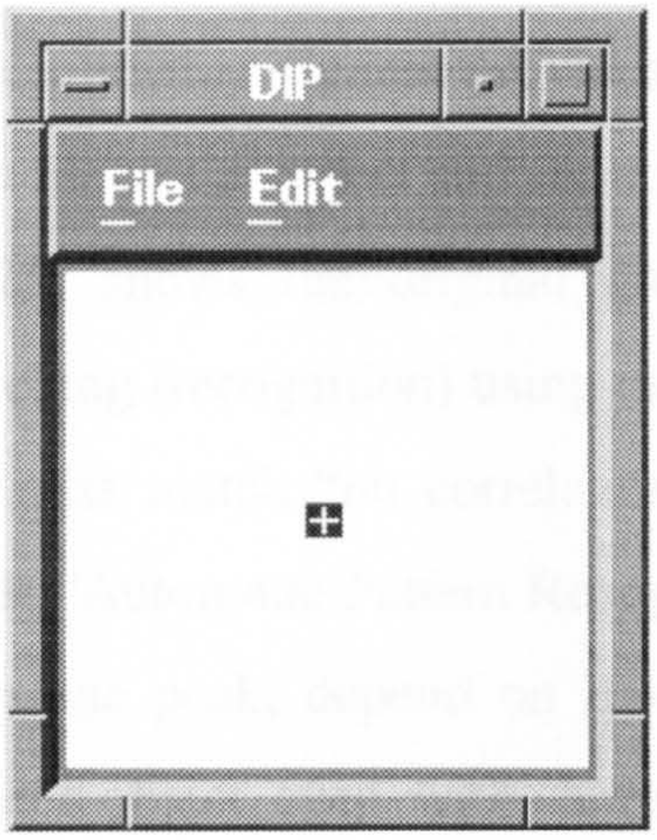
$$e = \left|(0-2)+(3-0)+(2-0)+(0-3)+(7-81)+((-7)-7)+(81-63)+(63-(-7))\right|$$
$$= \left|(-2)+3+2+(-3)+(-74)+(-14)+18+70\right|$$
$$= \left|0+0+88+(-88)\right|$$

$\therefore \quad e = 0$ (matching pattern)

We get the same result with the rest of all the different rotation Blocks of image 7 as shown above.

Solution to the Scaling Problem

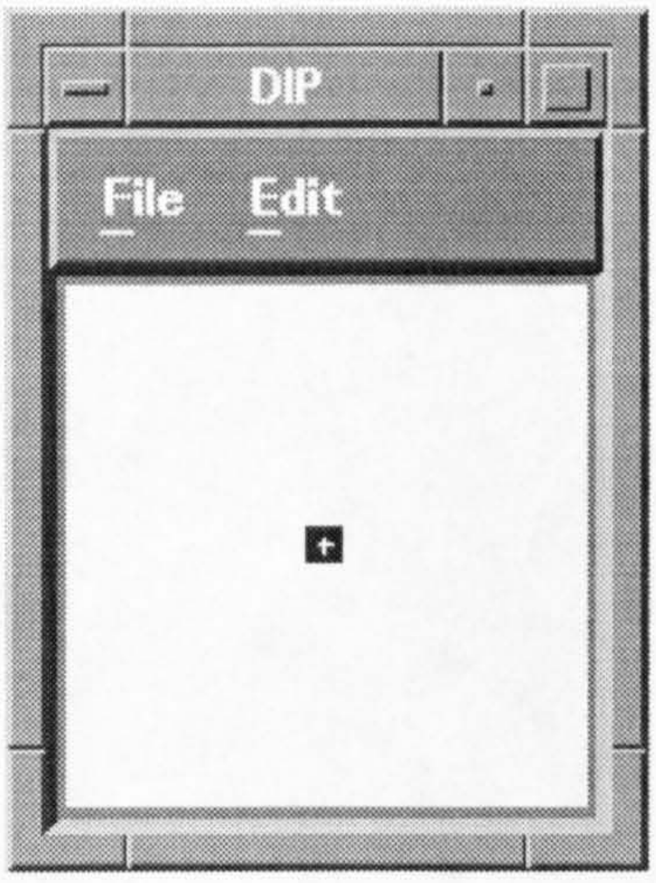
The implementations and results are shown in (**Fig 7.34**) and (**Fig 7.35**) Below.



111 111 111 255 111 111 111 111
111 111 111 255 111 111 111 111
111 111 111 255 111 111 111 111
255 255 255 255 255 255 255 255
111 111 111 255 111 111 111 111
111 111 111 255 111 111 111 111
111 111 111 255 111 111 111 111
111 111 111 255 111 111 111 111

0 0 0 66
0 0 3 39
0 0 0 39
0 0 0 3

Fig 7.34 A Block of image 8 **Data Block** **A Compressed Block**



111 111 111 111 111 111 111 111
111 111 111 111 111 111 111 111
111 111 111 255 111 111 111 111
111 255 255 255 255 255 111 111
111 111 111 255 111 111 111 111
111 111 111 255 111 111 111 111
111 111 111 111 111 111 111 111
111 111 111 111 111 111 111 111

0 0 0 44
0 0 3 35
0 0 0 35
0 0 0 17

Fig 7.35 A Block of image7 **Data Block** **A Compressed Block**

Applying the matching equation

$$e = \left|(0-0)+(3-3)+(0-0)+(0-0)+(66-44)+(35-39)+(35-39)+(3-17)\right|$$

$$= \left|22+(-4)+(-4)+(-14)\right|$$

$$\hat{e} = \left|22+(-22)\right|$$

$$\therefore \ \hat{e} = 0 \ (\text{matching pattern})$$

The following figures provides much cleaner results, than by displaying the data file of the object and the compressed data file. The figures (**Fig 8.1 to Fig 14.25**) shows the original greyscales images but in 3D space view, the matching (recognition) using the Cross-Correlation method with a single peak, the miss match "no correlation" (i.e. no peak), and the matching using the **APR** "Automatic Pattern Recognition method" respectively, with one or more than one peak, depend on if there are more than one similar object to the located object, even with variant rotation angle.

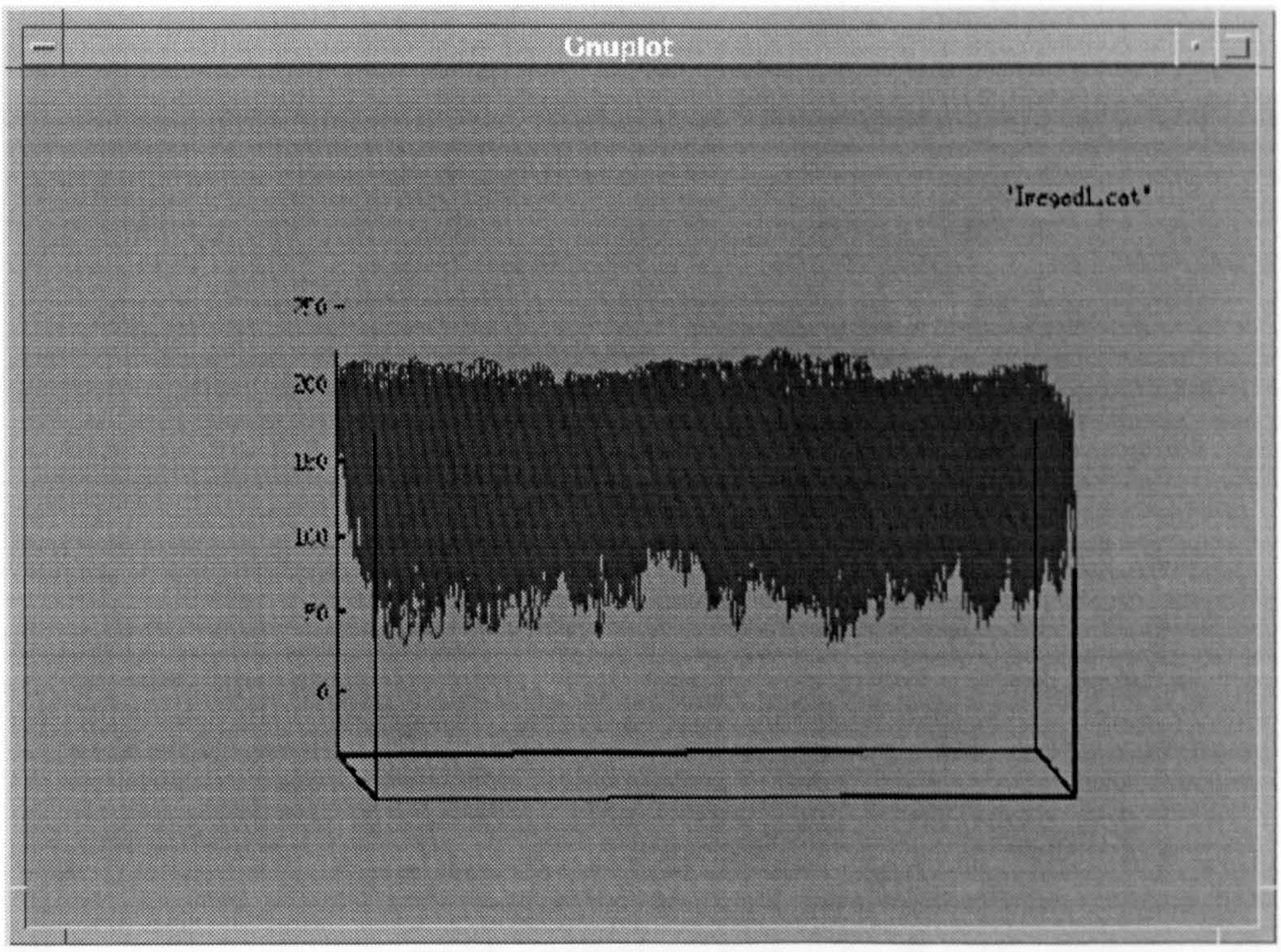


Fig 8.1 The Original image

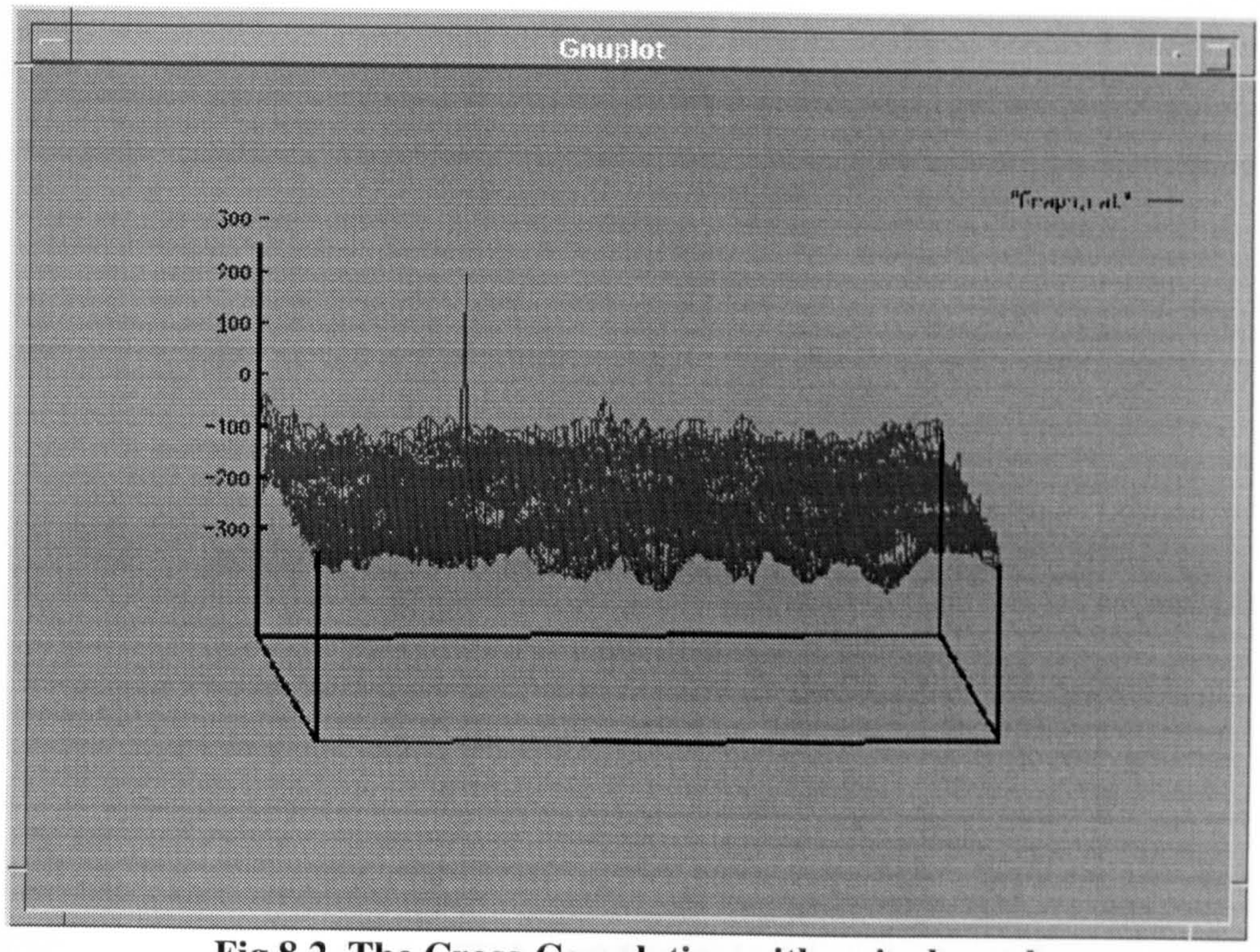


Fig 8.2. The Cross-Correlation with a single peak

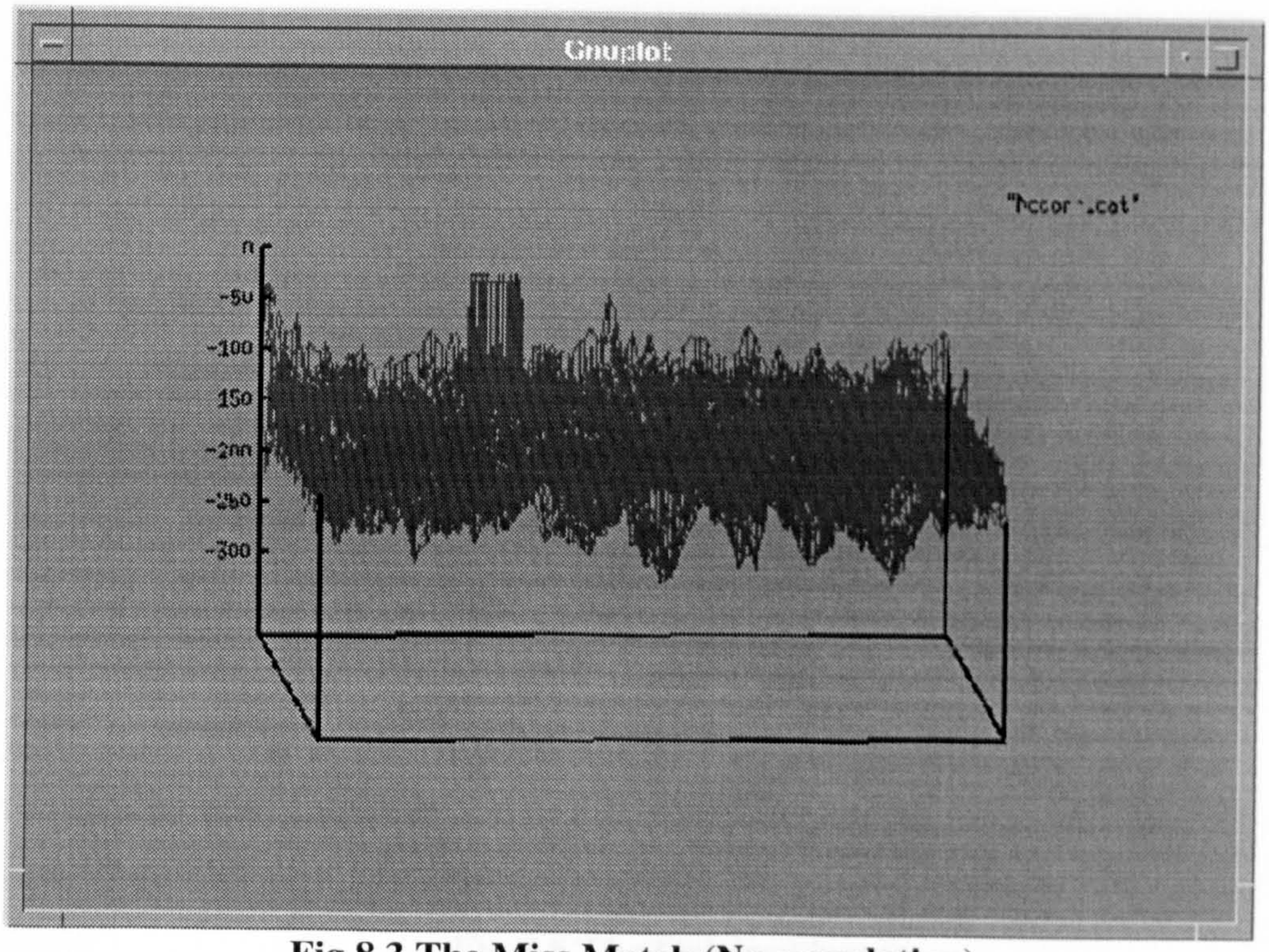


Fig 8.3 The Miss Match (No correlation)

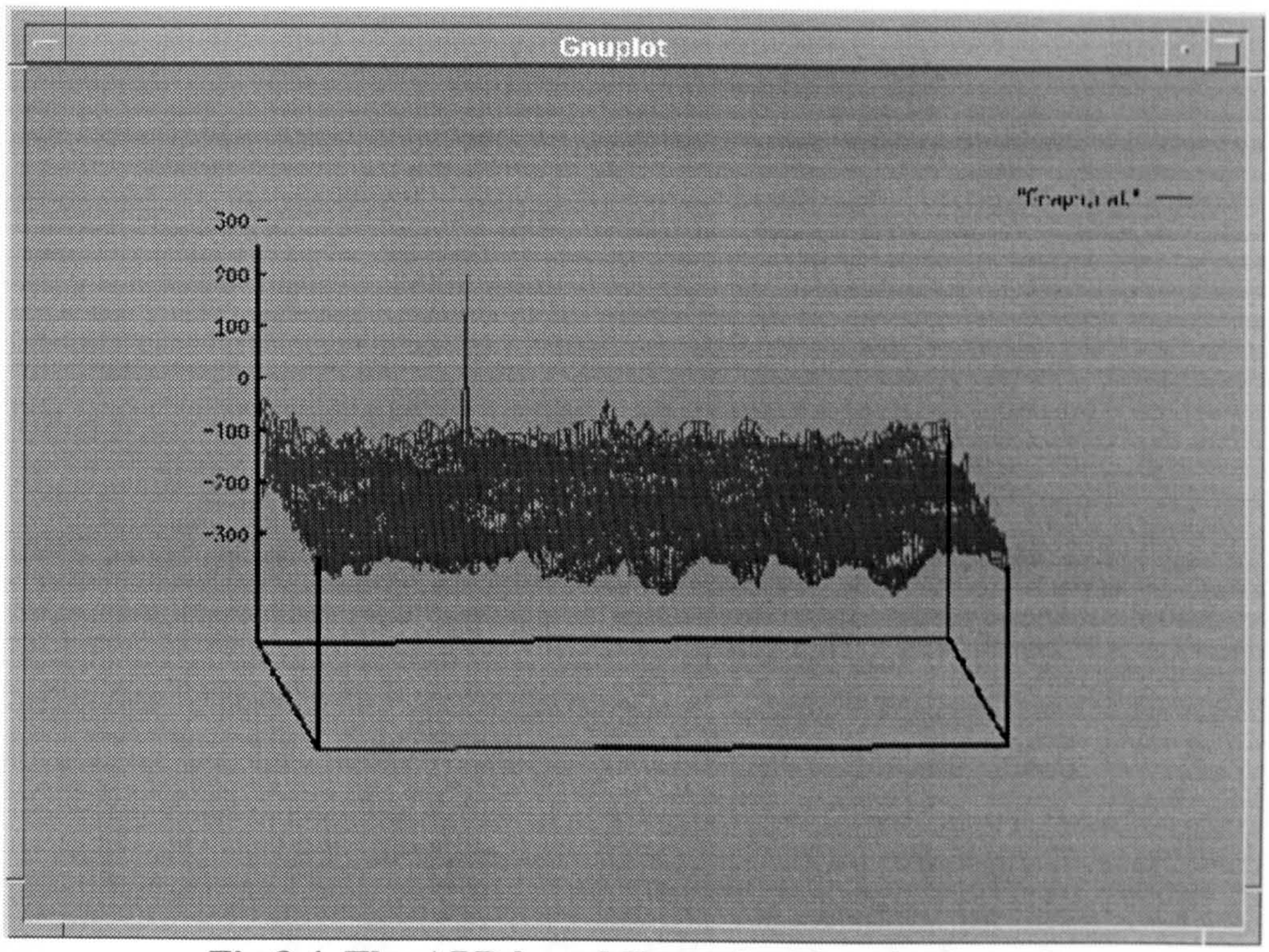


Fig 8.4. The APR based Fractal with a single peak

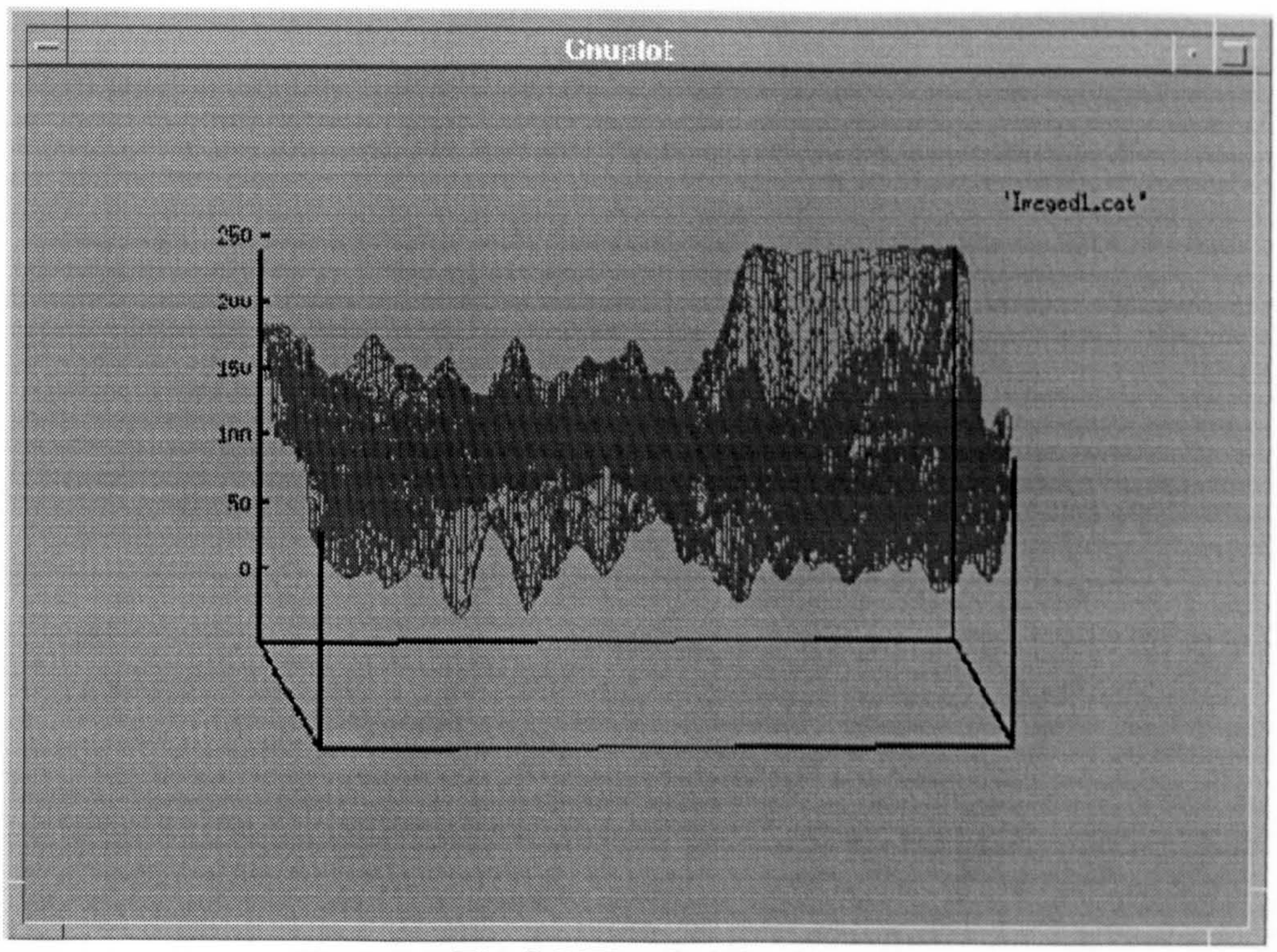


Fig 8.5 The Original image

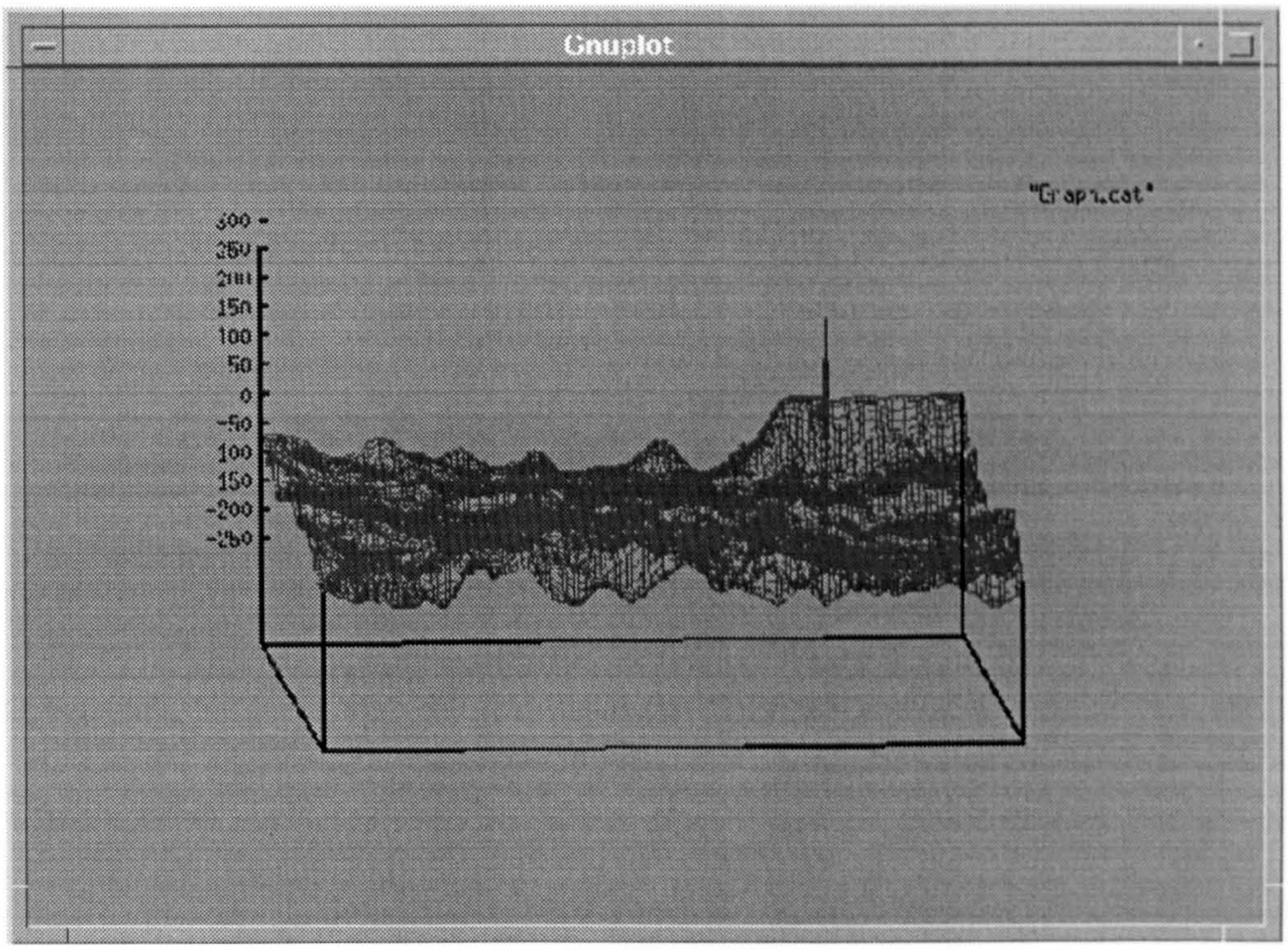


Fig 8.6. The Cross-Correlation with a single peak

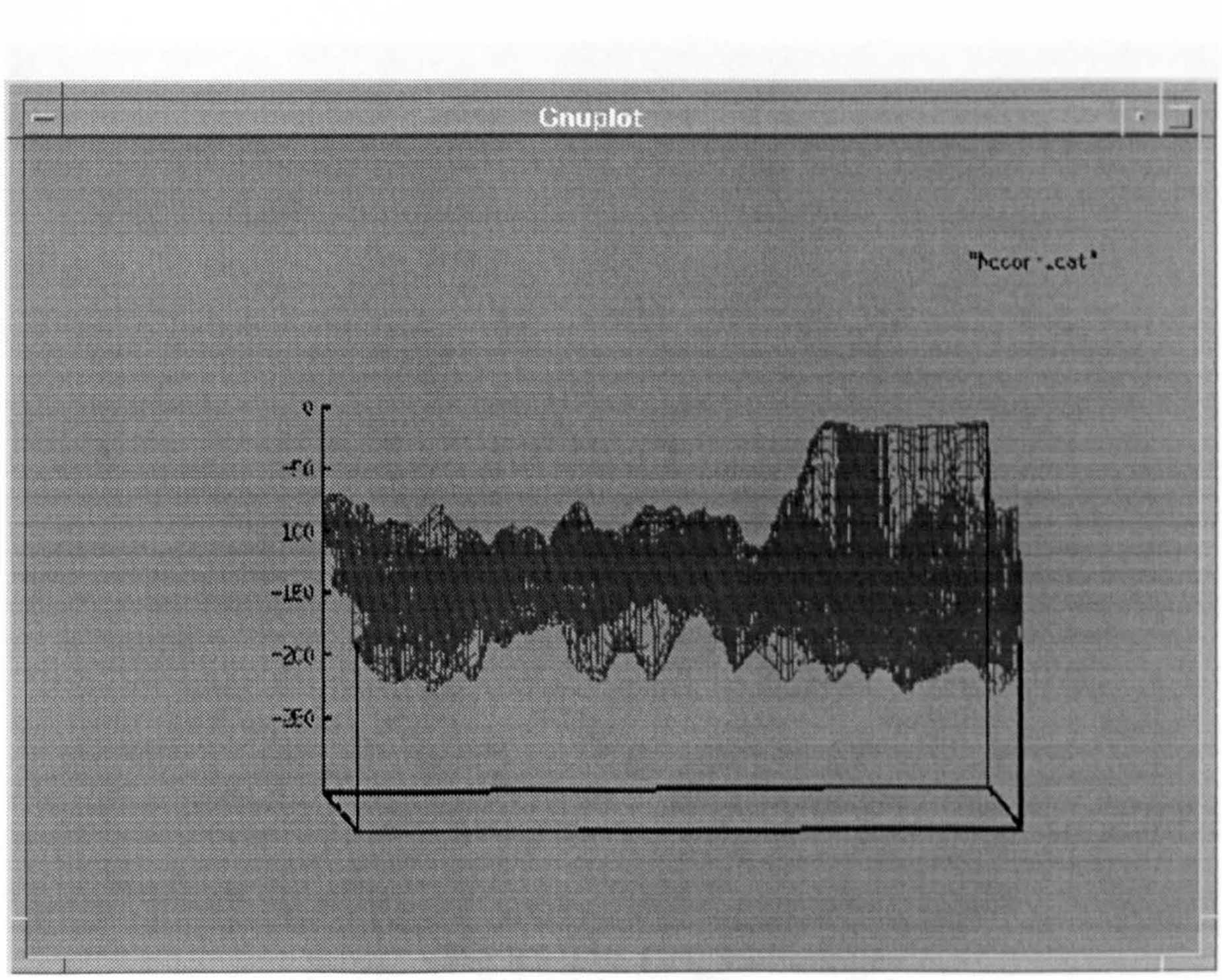


Fig 8.7 The Miss Match (No correlation)

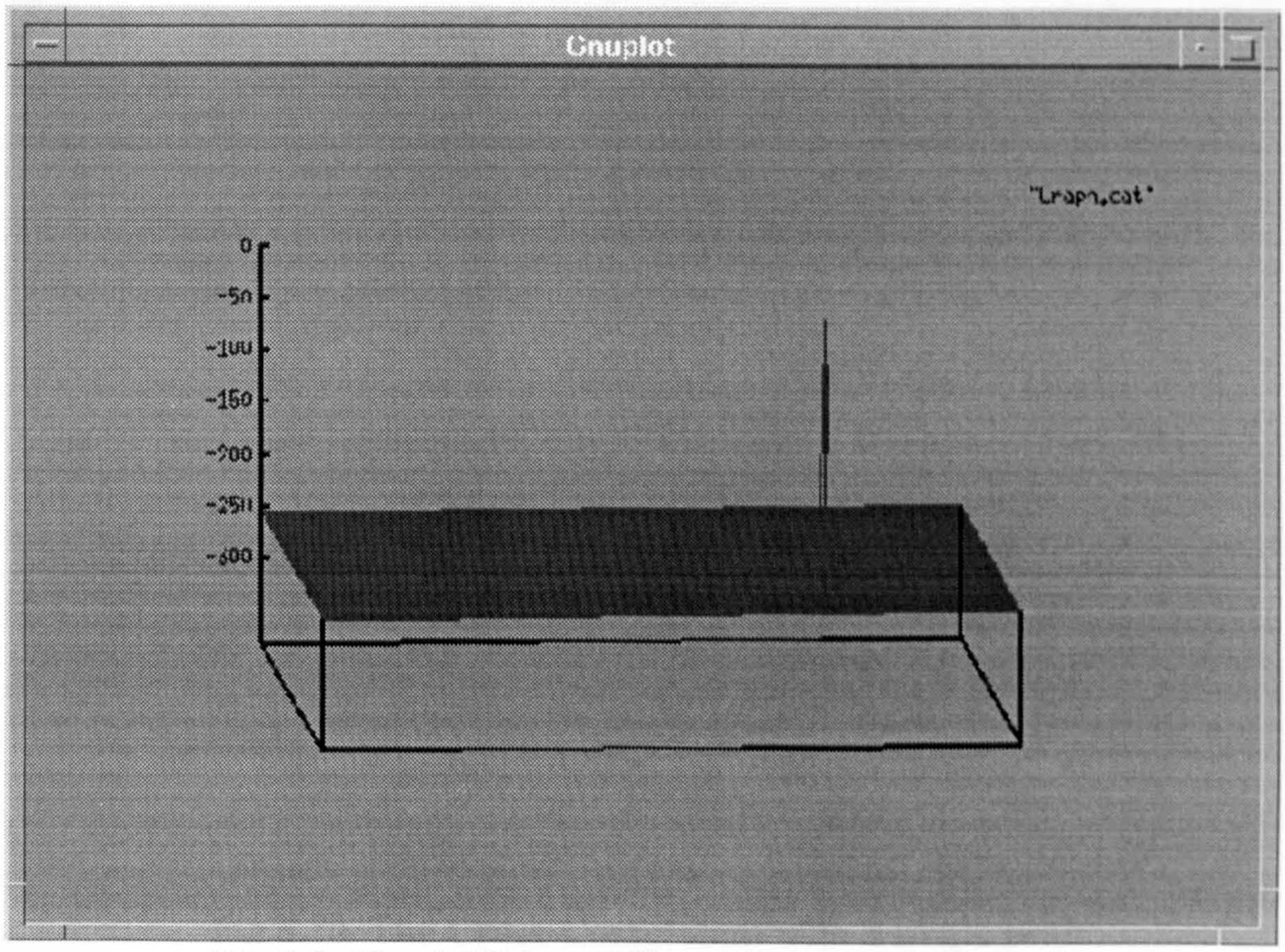


Fig 8.8. The APR based Fractal with a single peak

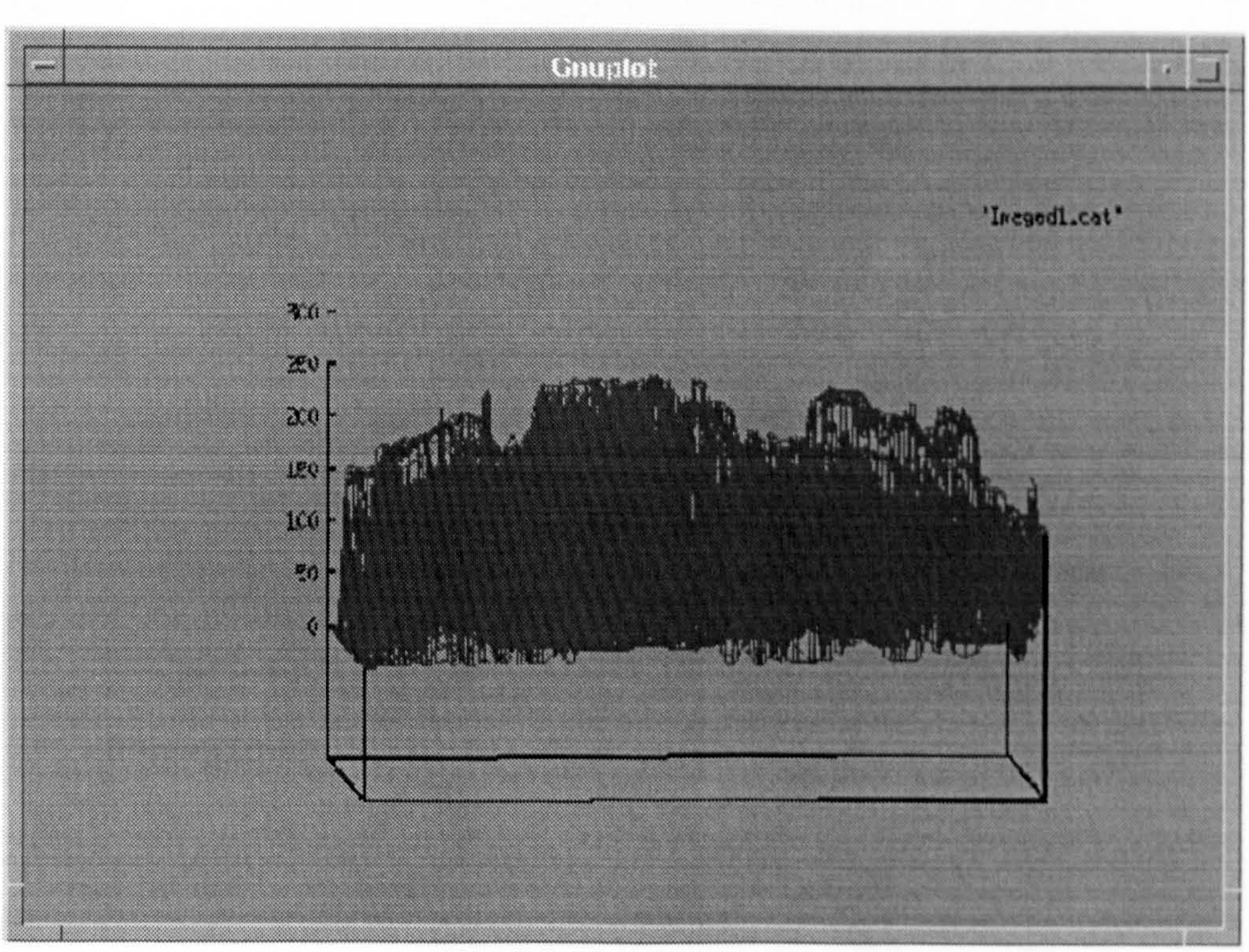


Fig 8.9 The Original image

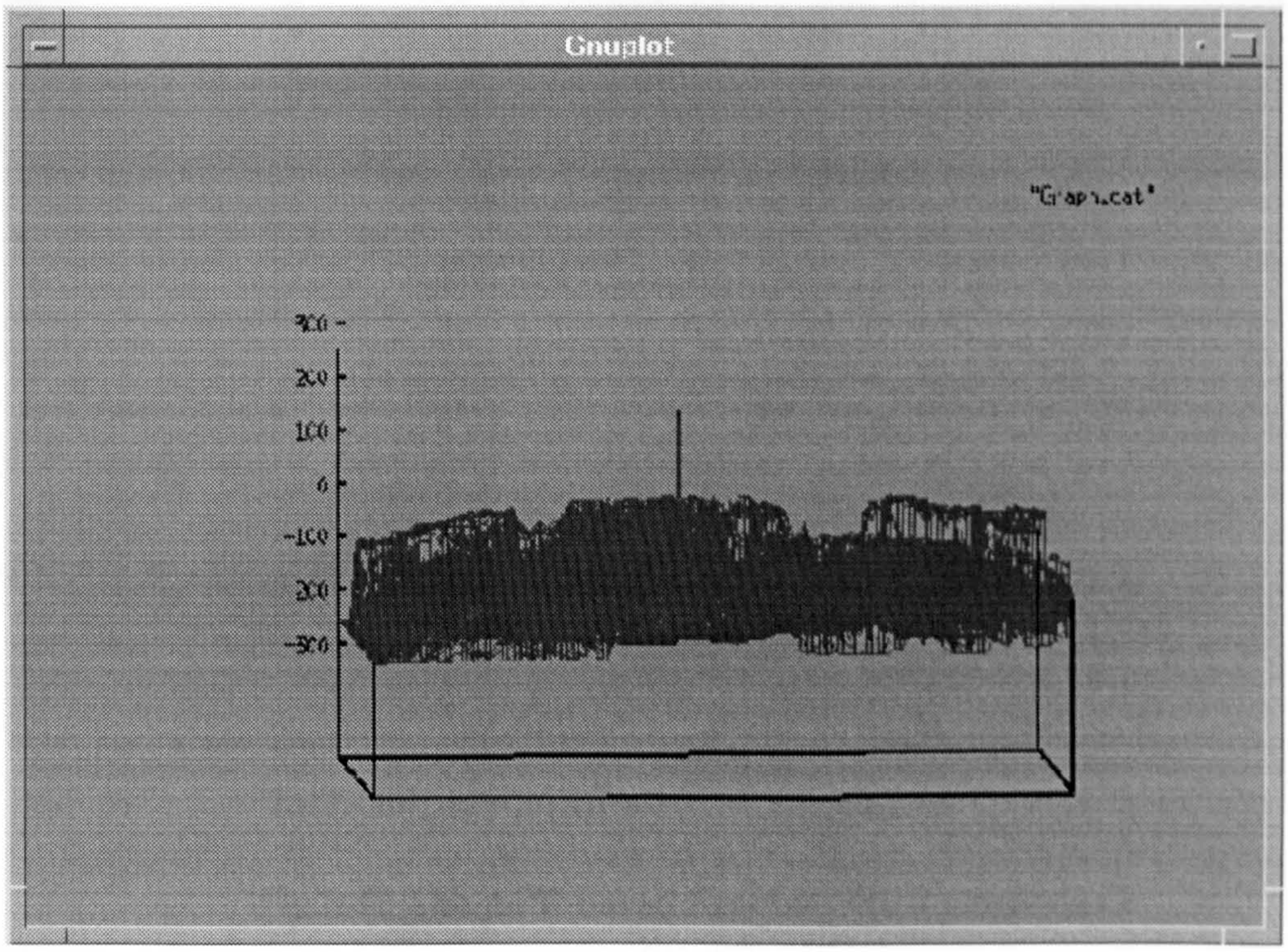


Fig 8.10. The Cross-Correlation with a single peak

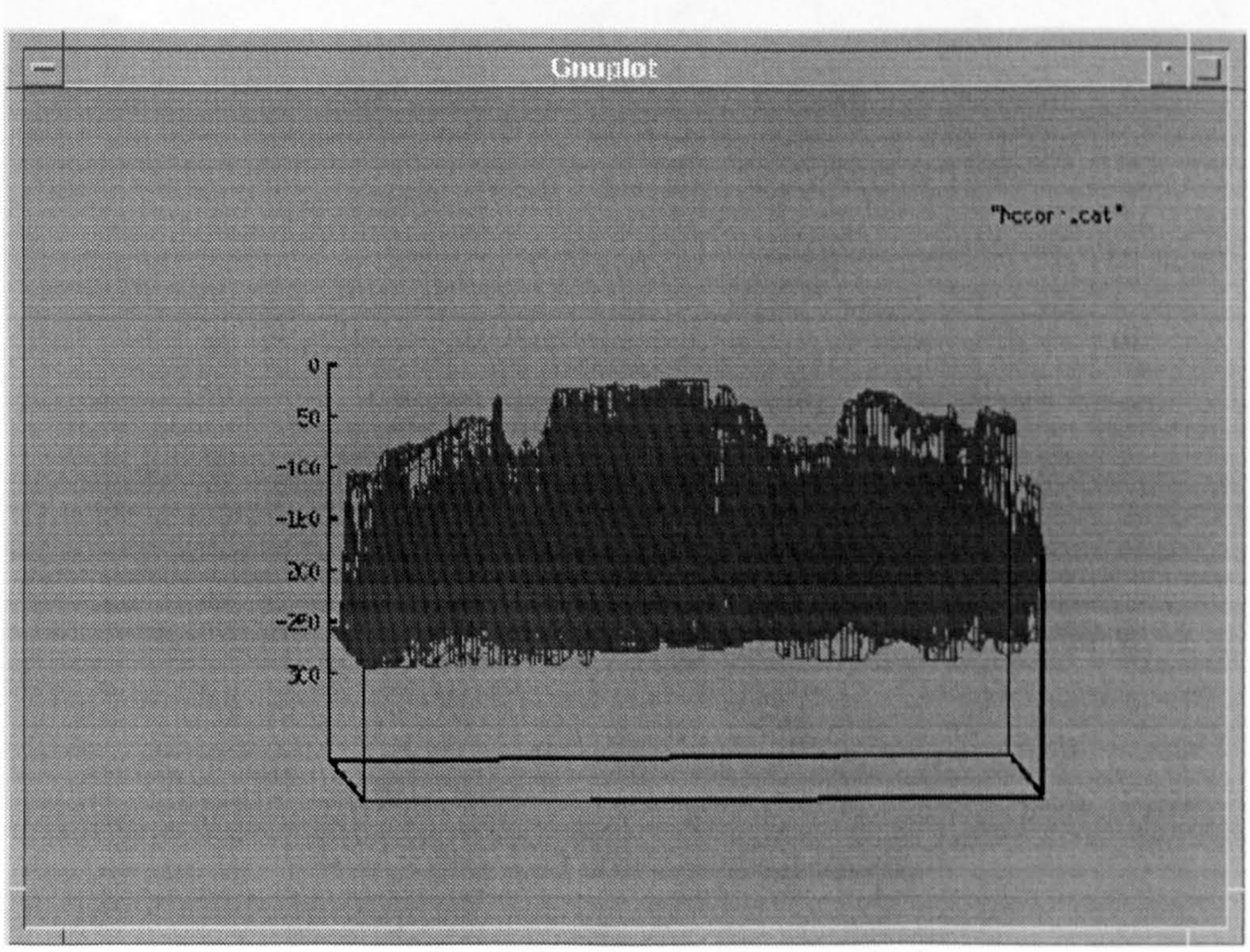


Fig 8.11 The Miss Match (No correlation)

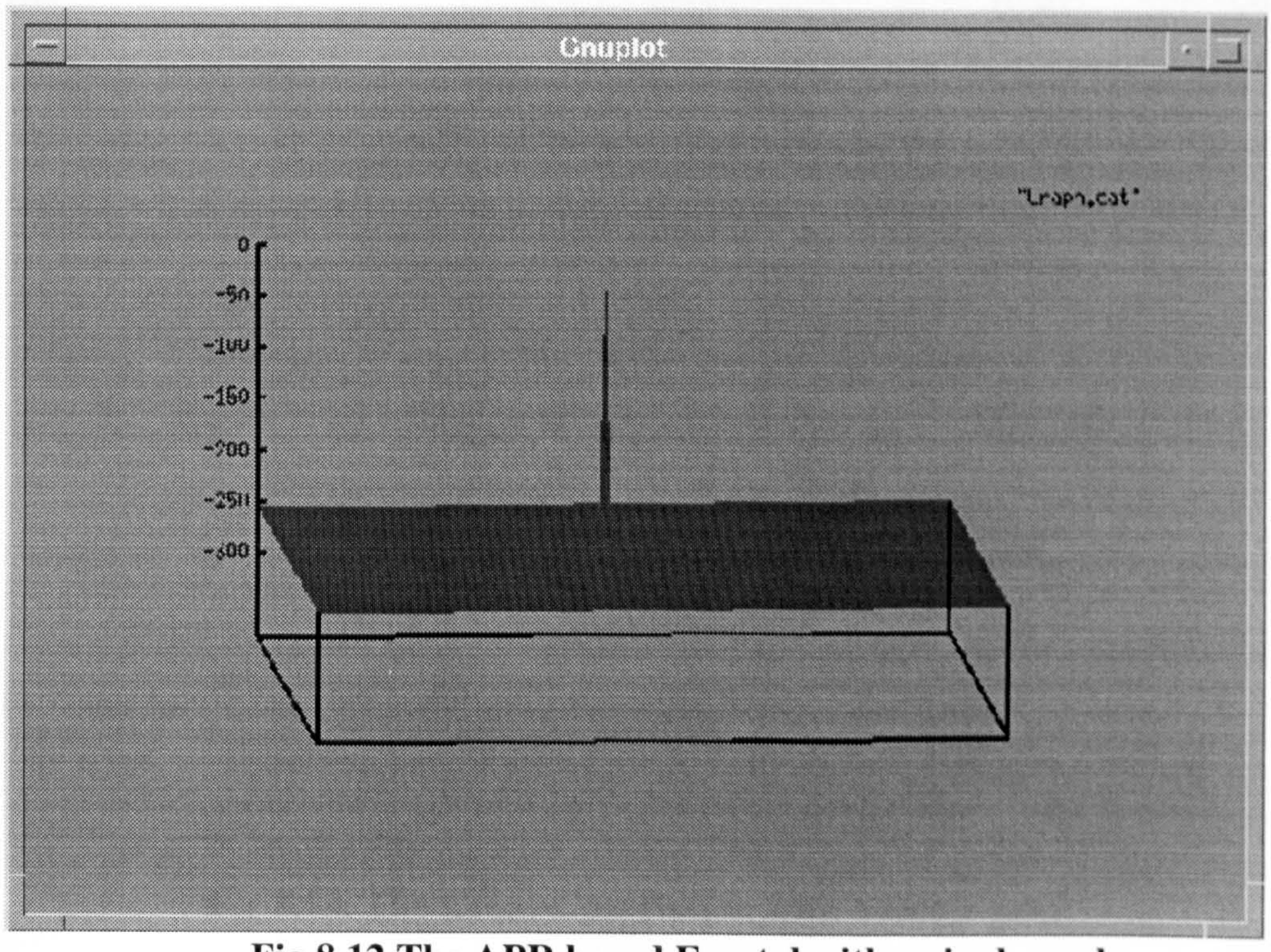


Fig 8.12 The APR based Fractal with a single peak

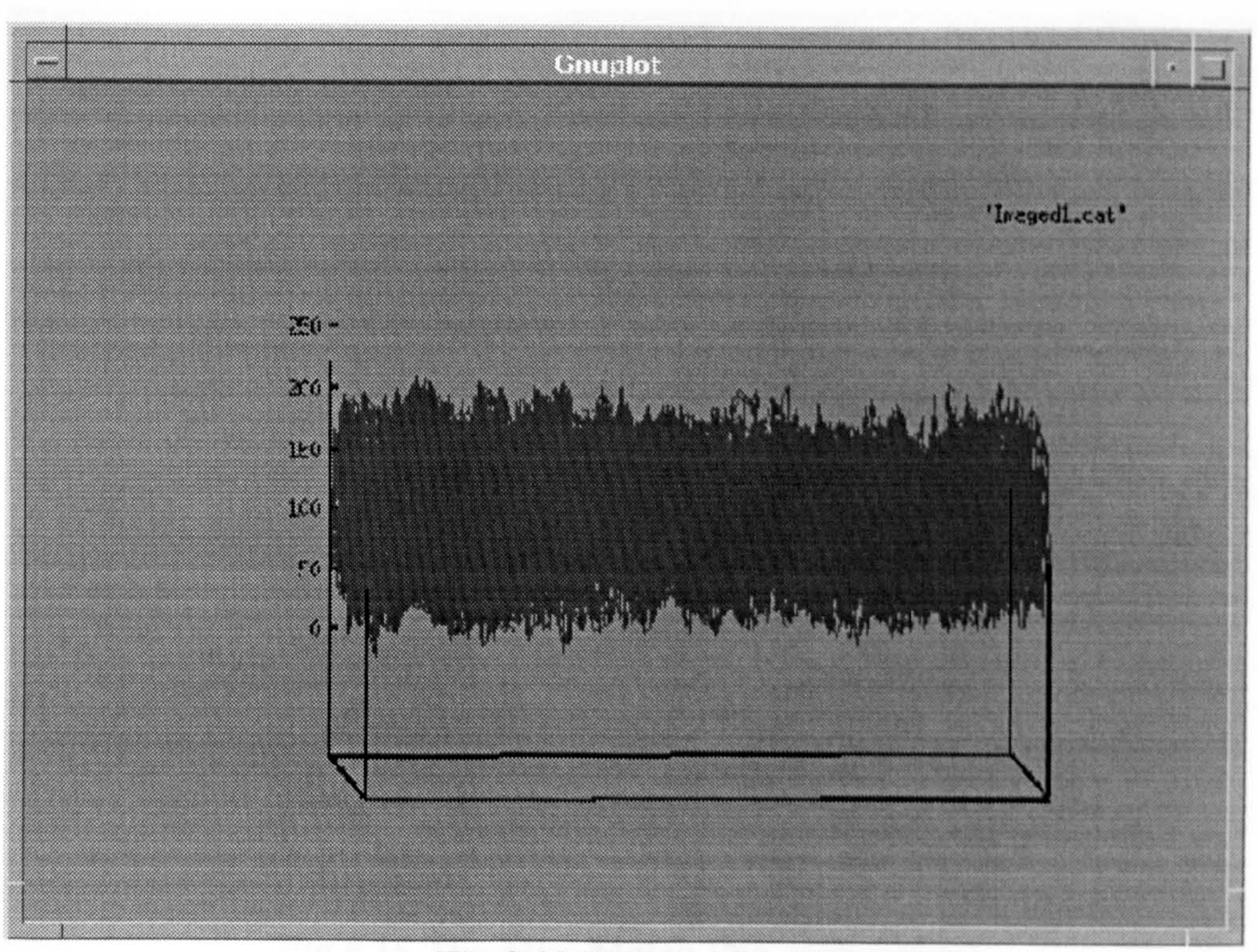


Fig 8.13 The Original image

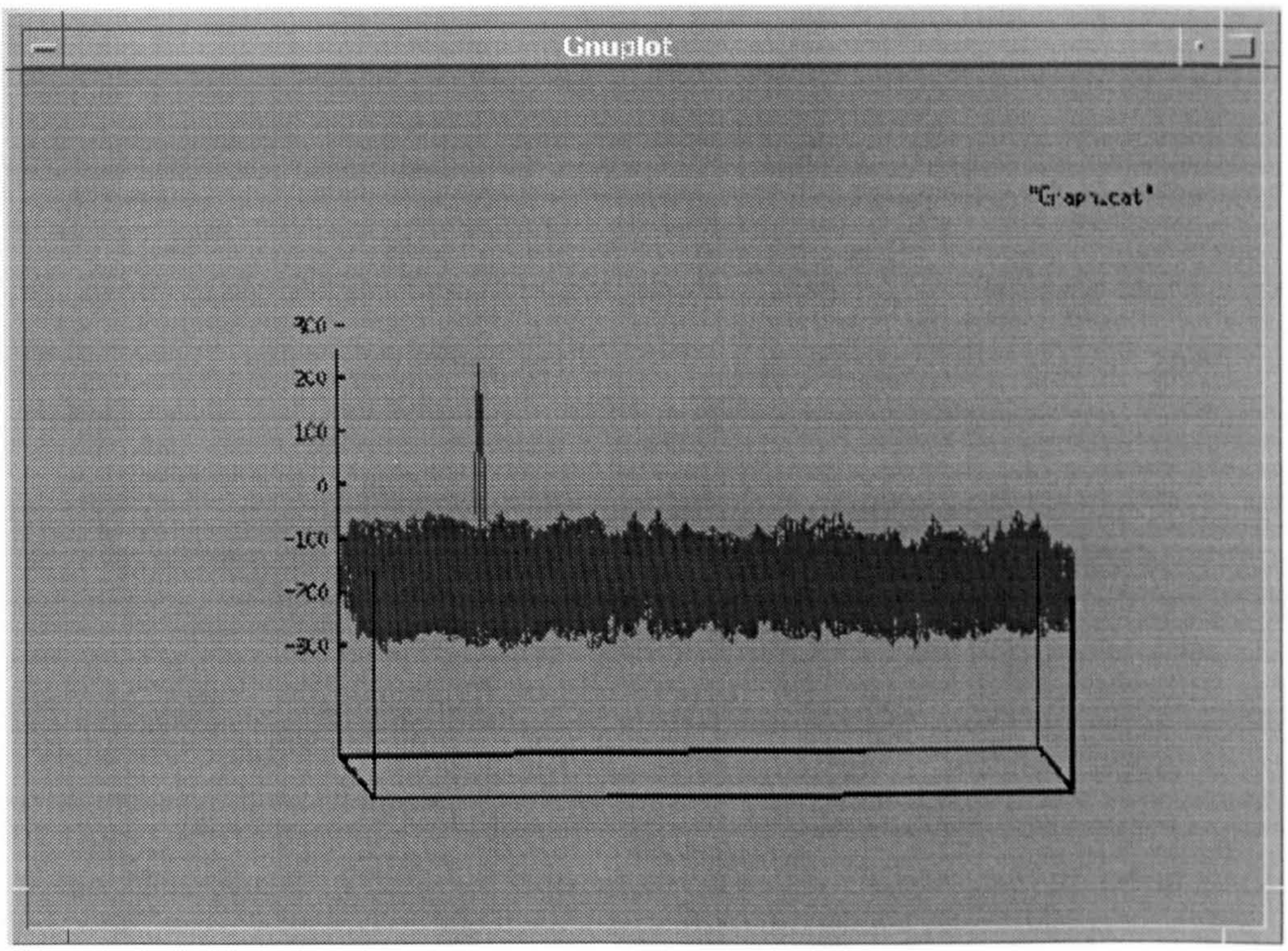


Fig 8.14. The Cross-Correlation with a single peak

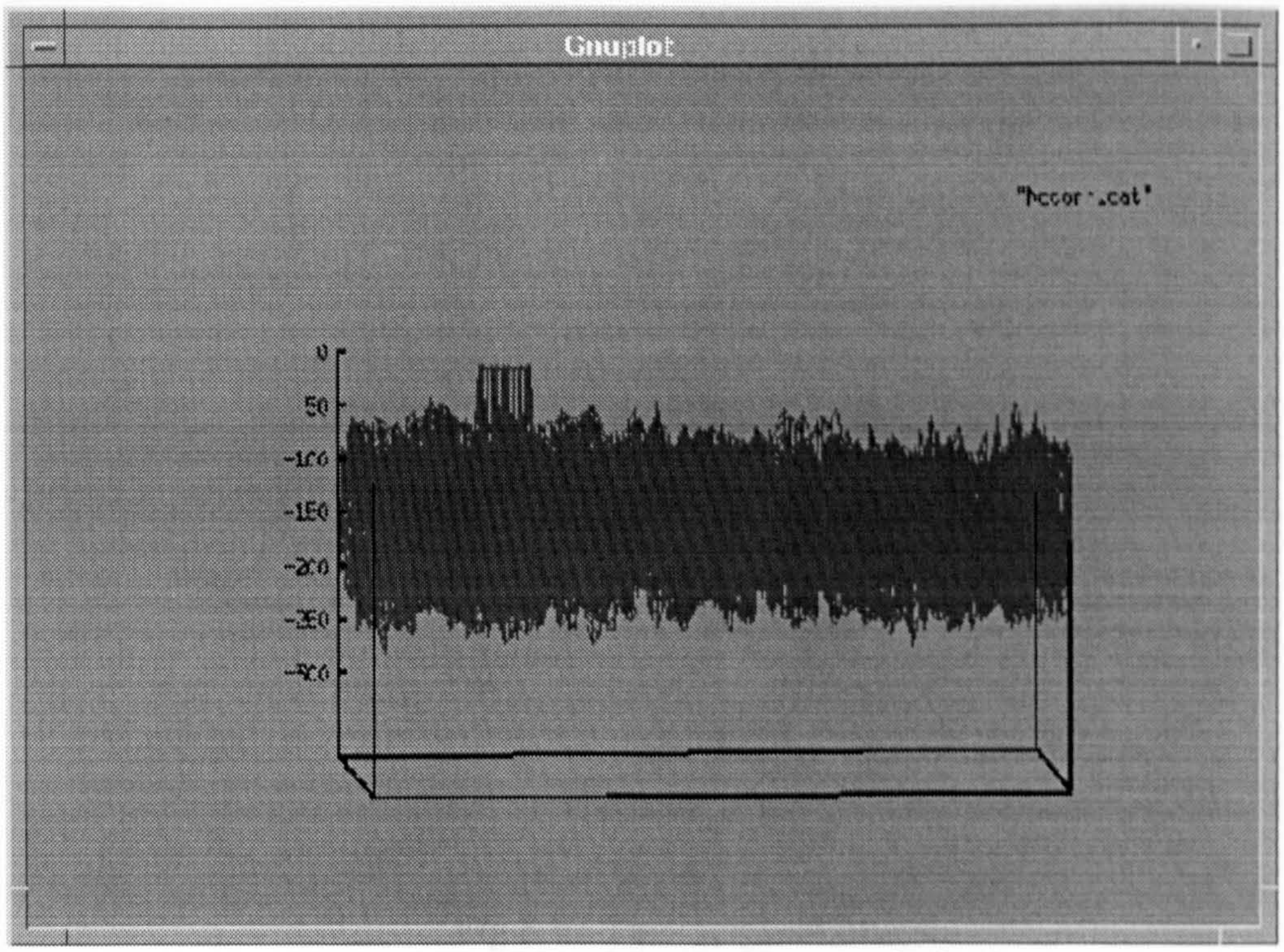


Fig 8.15 The Miss Match (No correlation)

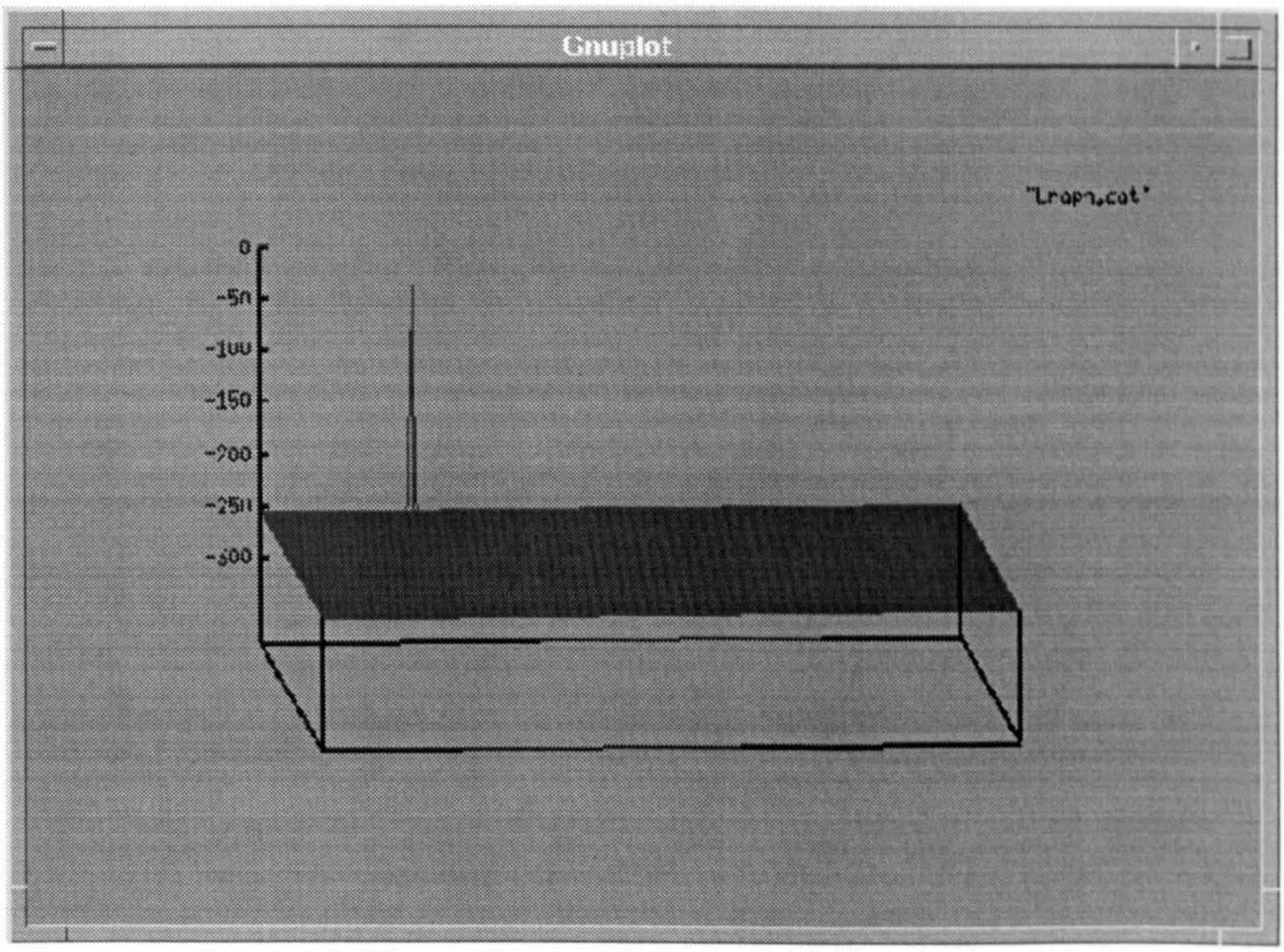


Fig 8.16. The APR based Fractal with a single peak

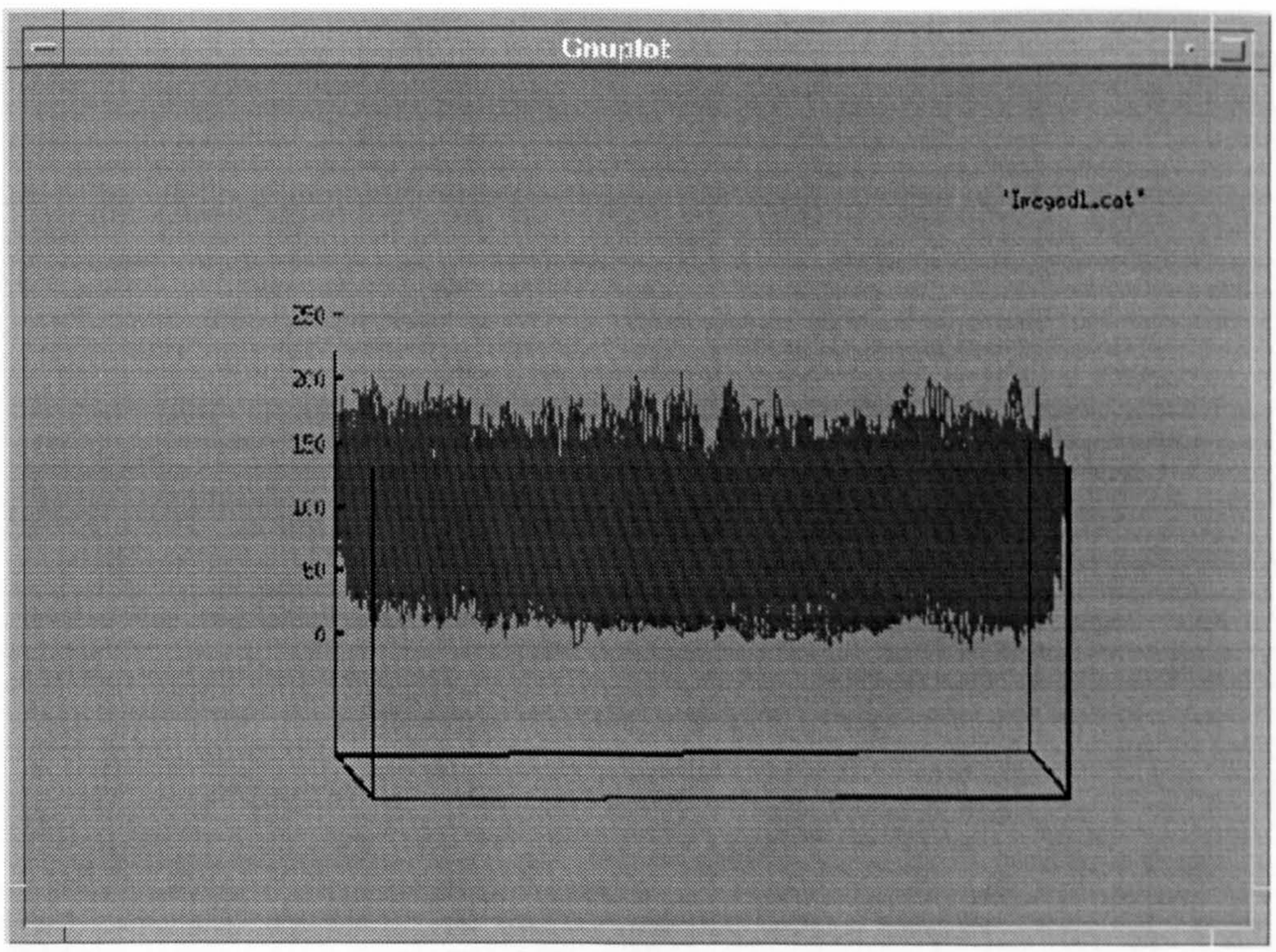


Fig 8.17 The Original image

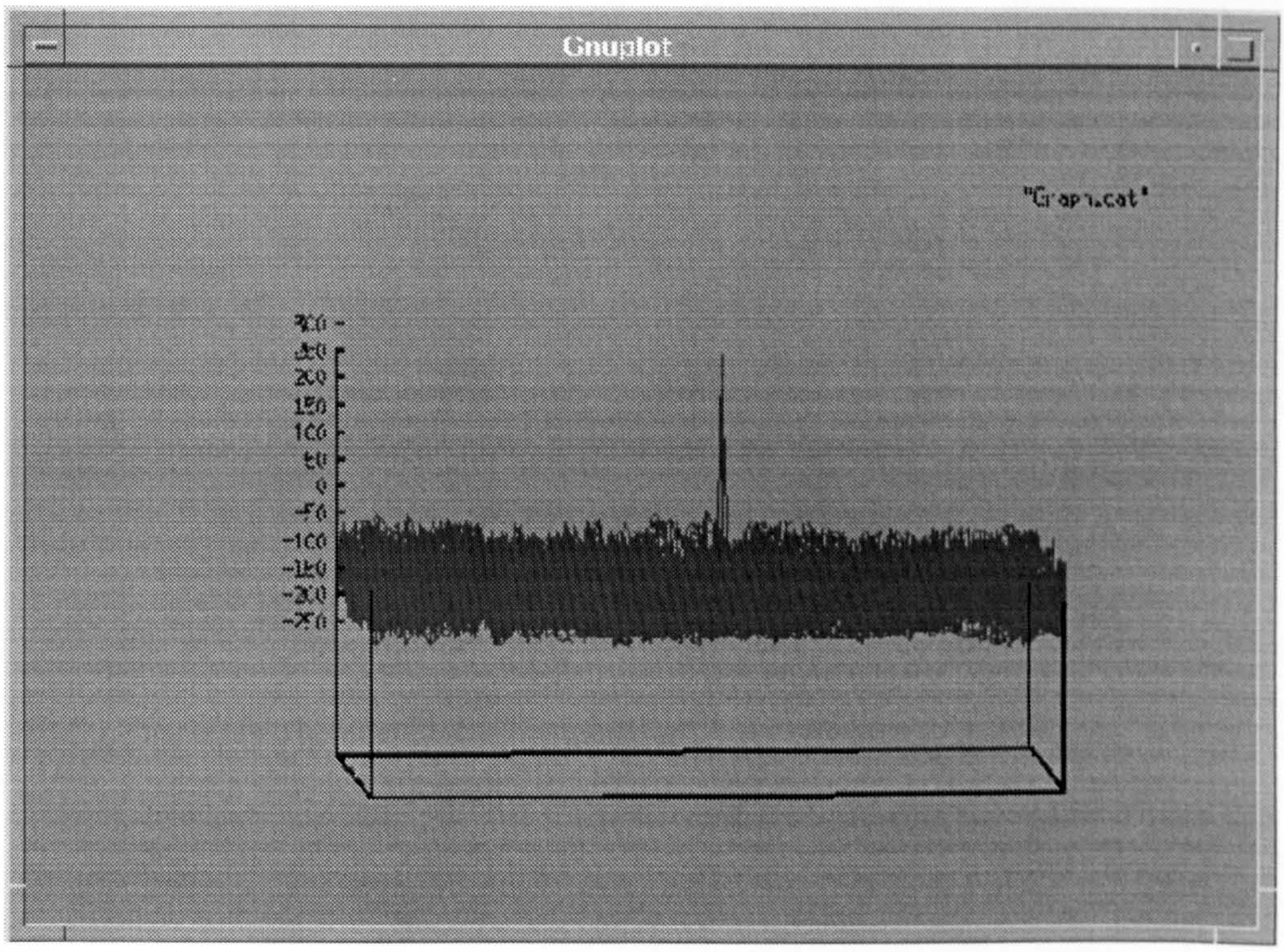


Fig 8.18. The Cross-Correlation with a single peak

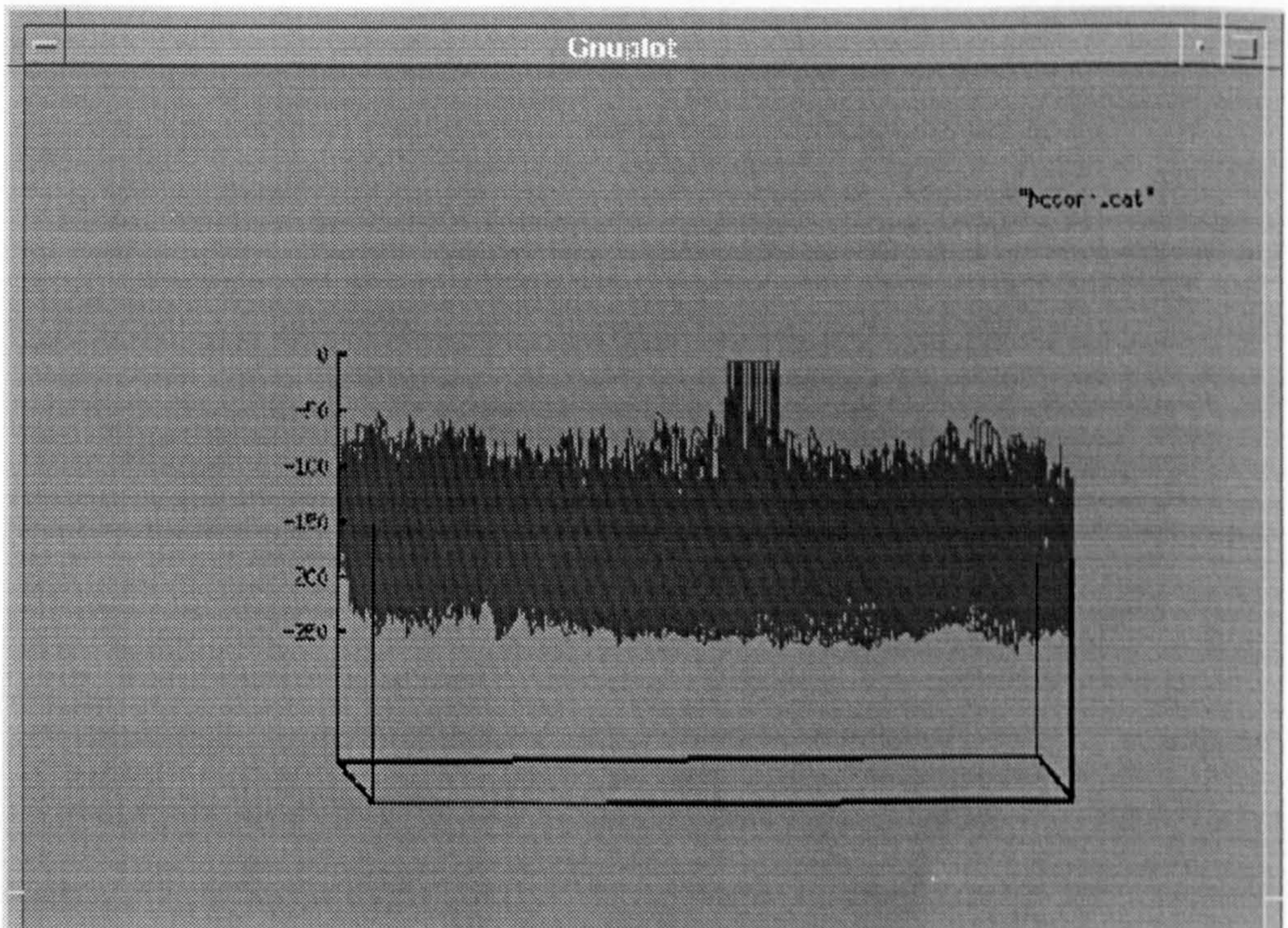


Fig 8.19 The Miss Match (No correlation)

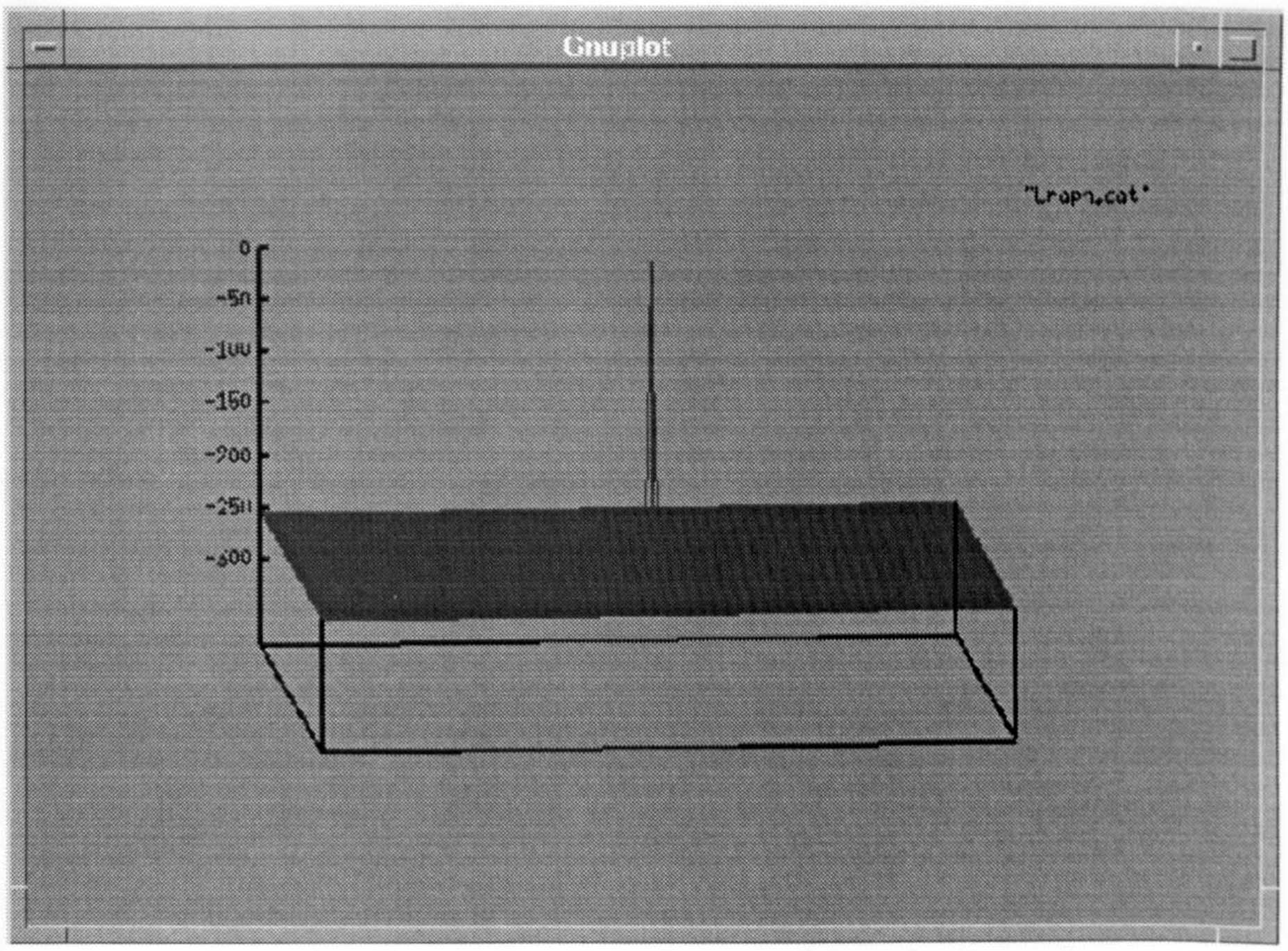


Fig 8.20. The APR based Fractal with a single peak

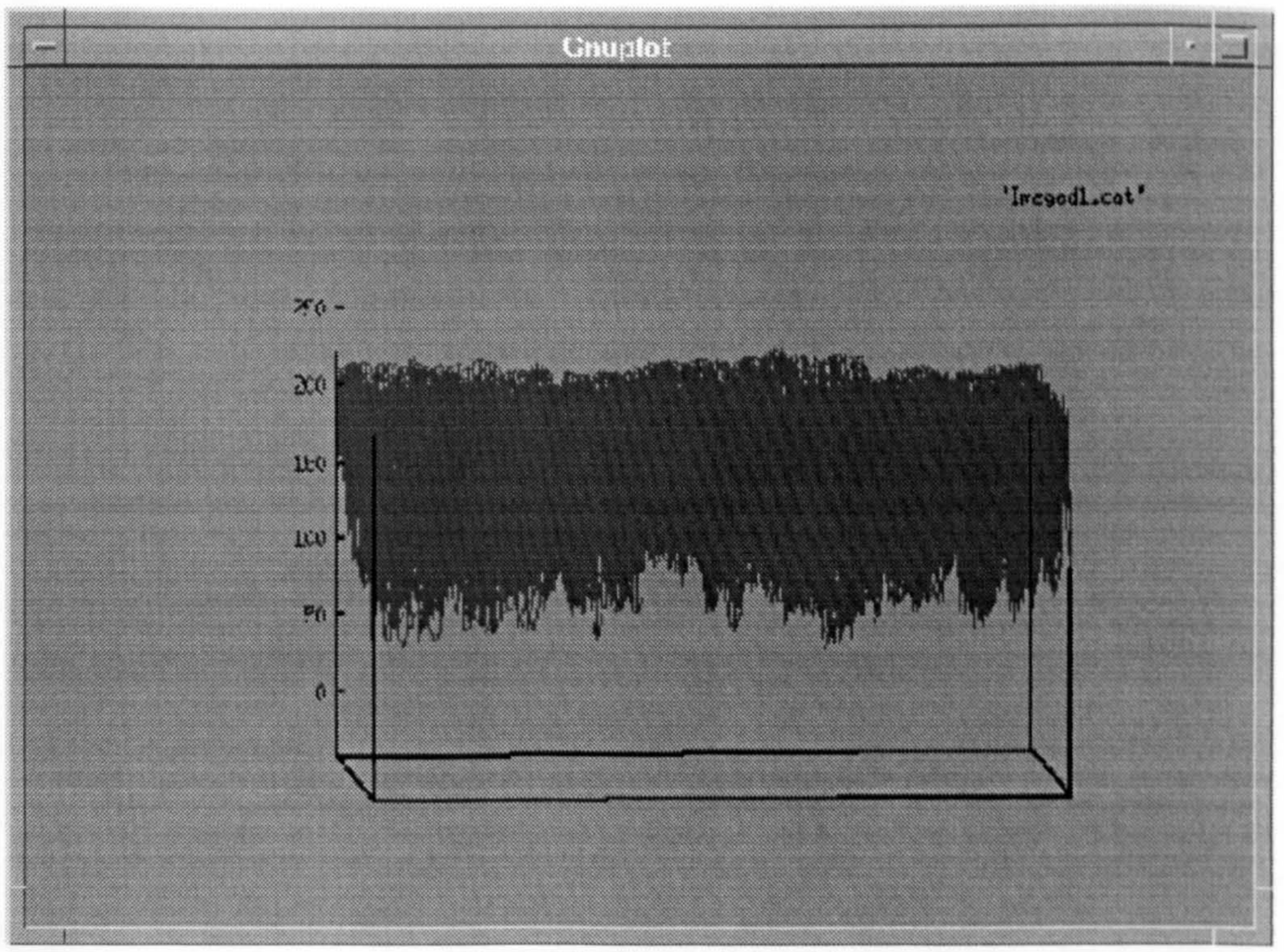


Fig 8.21 The Original image

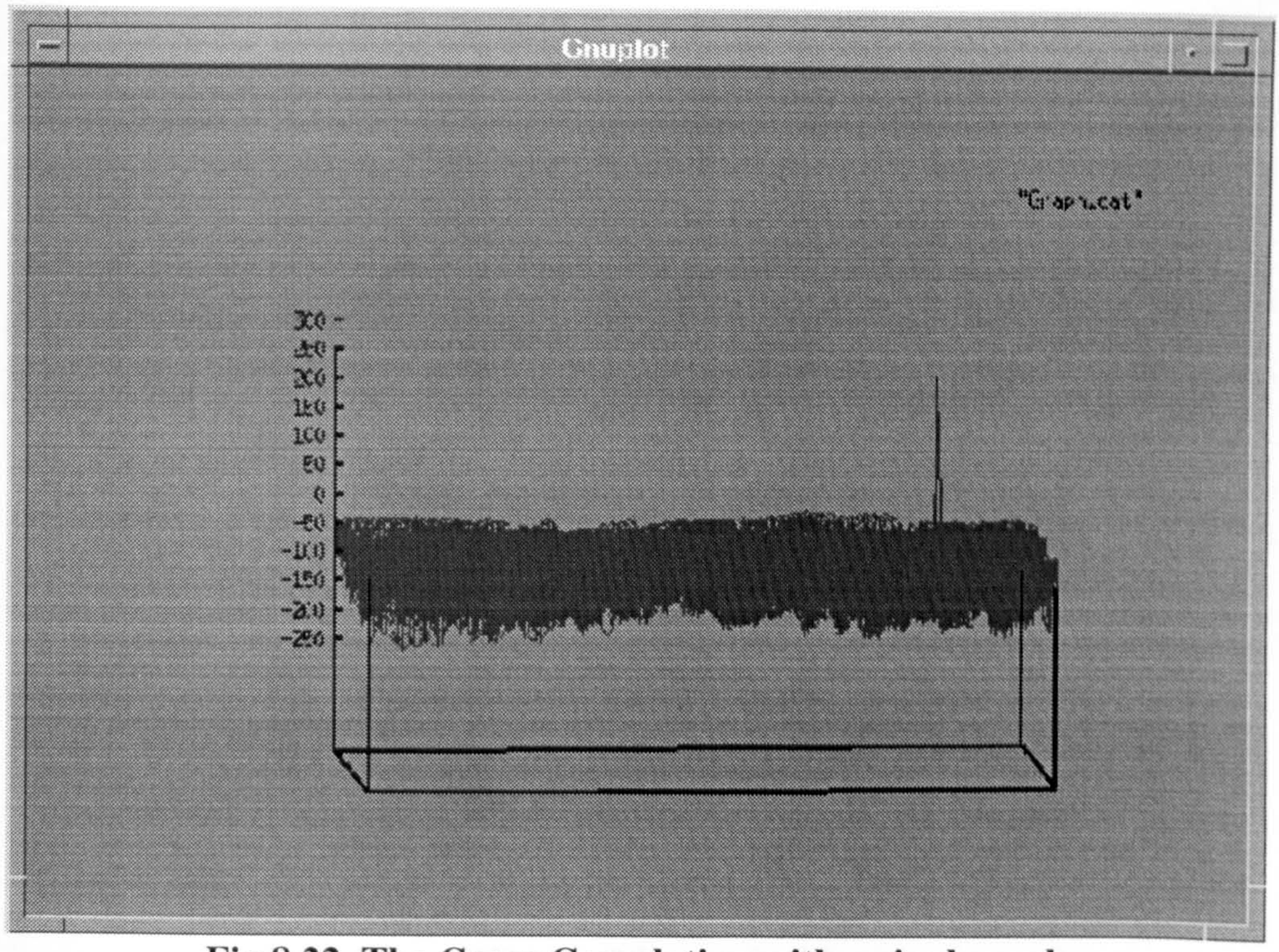


Fig 8.22. The Cross-Correlation with a single peak

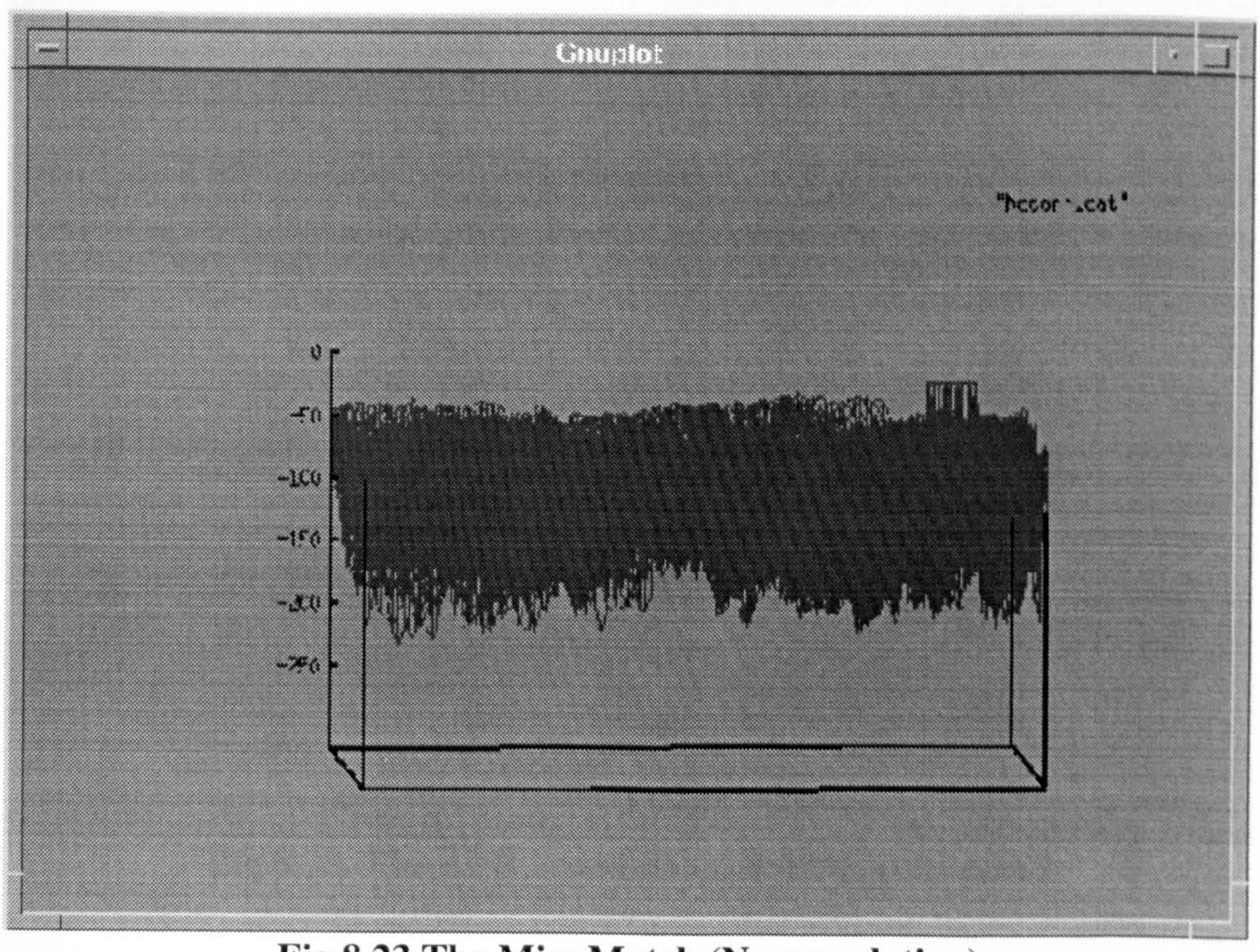


Fig 8.23 The Miss Match (No correlation)

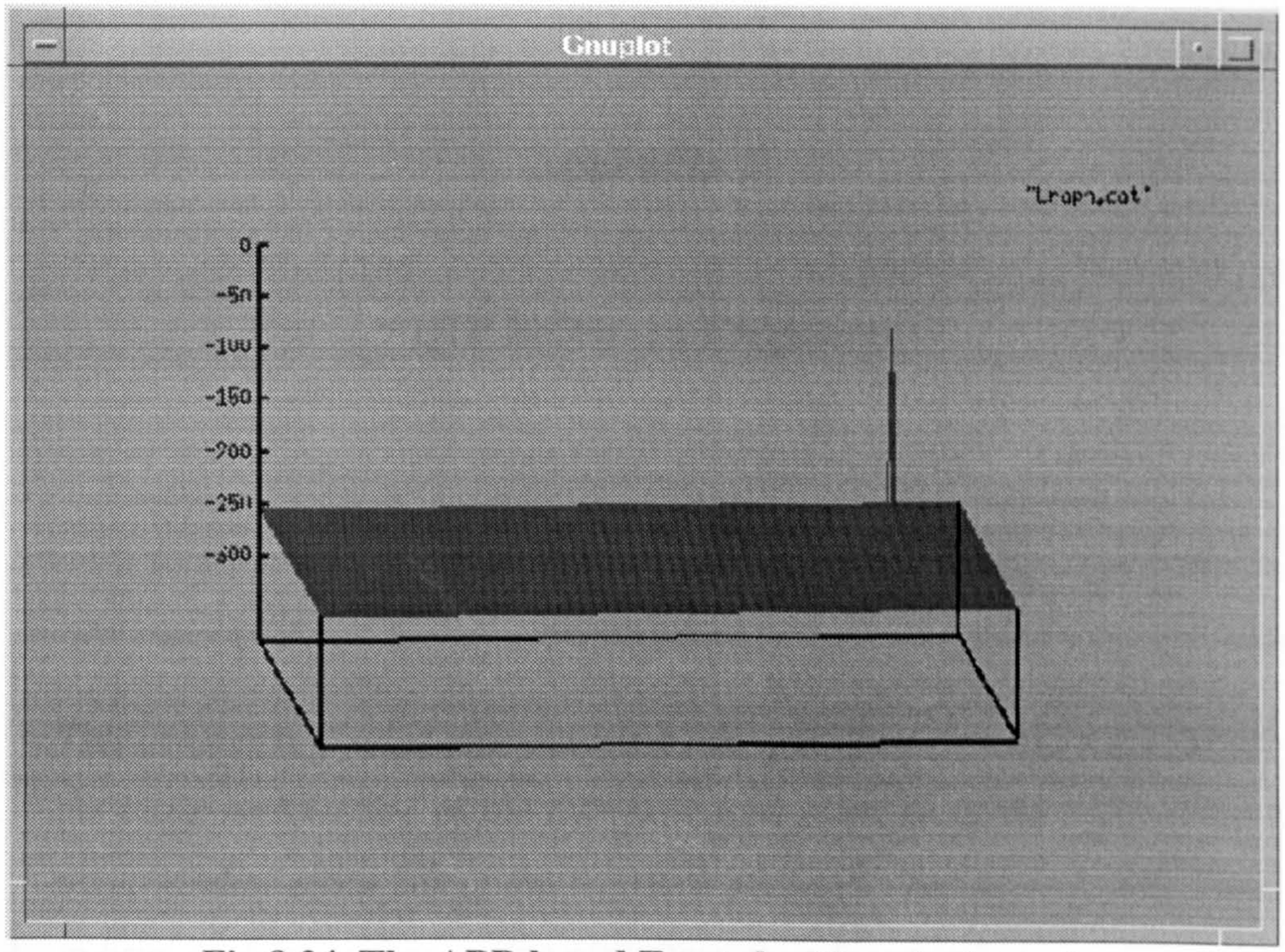


Fig 8.24. The APR based Fractal with a single peak

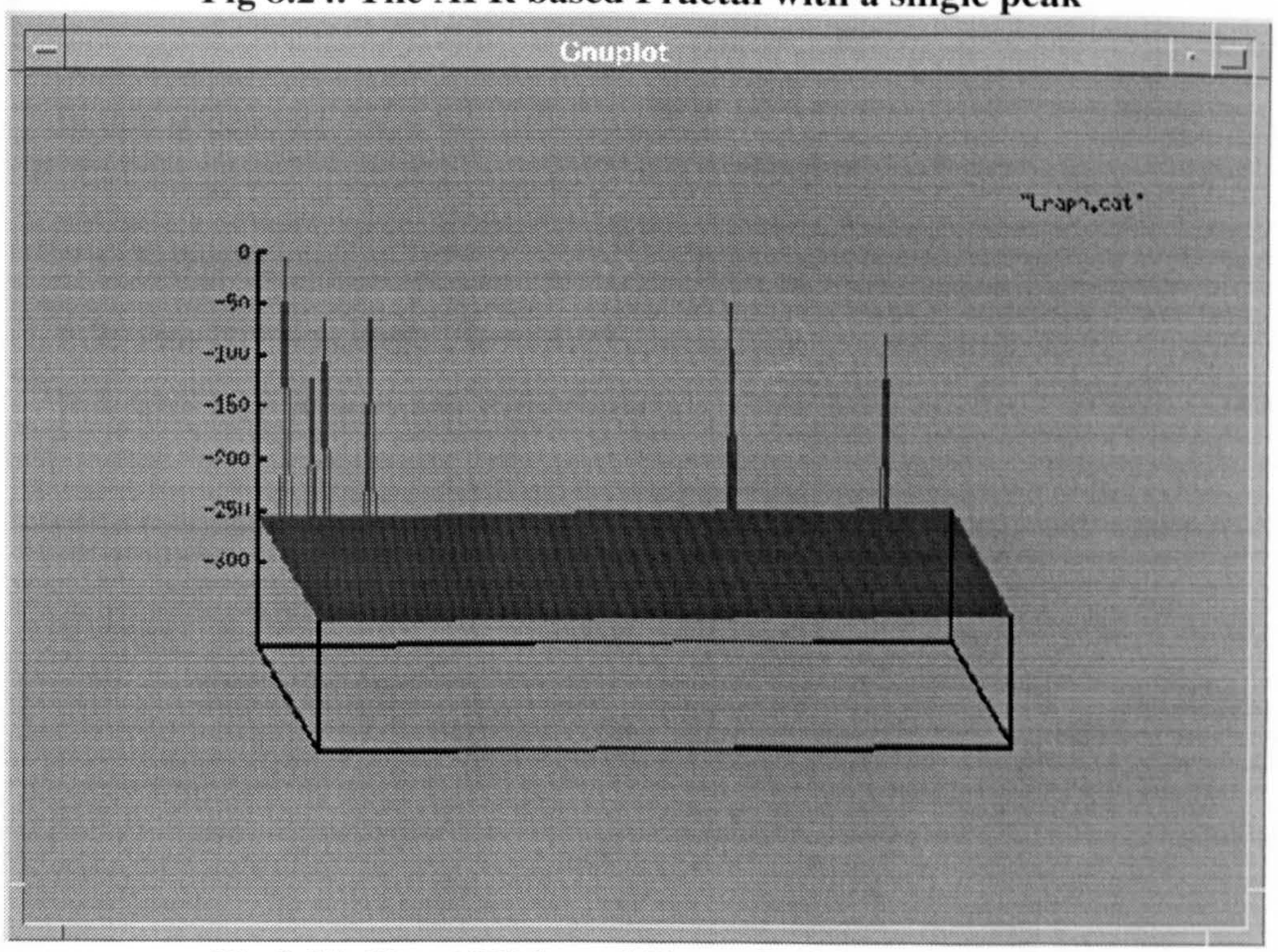


Fig 8.25. The APR based Fractal with multi peaks

Appendix F

DIP Software Engineer

This chapter is divided into two main parts, the first part provides details of the basic architecture of the DIP flowchart package in the order as in the main window menu bar of the interface package. The second part explains in brief each image processing algorithm main functions..

In this appendix: **Page No**

F.1. Introduction

The flowchart of the DIP software package which is divided into subsections provides details of all the operation processes of all the programs, all can be found in **section C.2**. Each subsection reflects the interface of each option in the main window menu bar. Section C.3 contains the functions and descriptions of all the DIP Image Processing algorithms that the library is built of. For convenience, the functions are arranged in alphabetical order. Here is a sample layout so the reader can easily understand the format of the lookup;
Note: only the header functions are listed, not the body of the function. All the software functions of the DIP package can be found on a floppy disk included with this thesis.

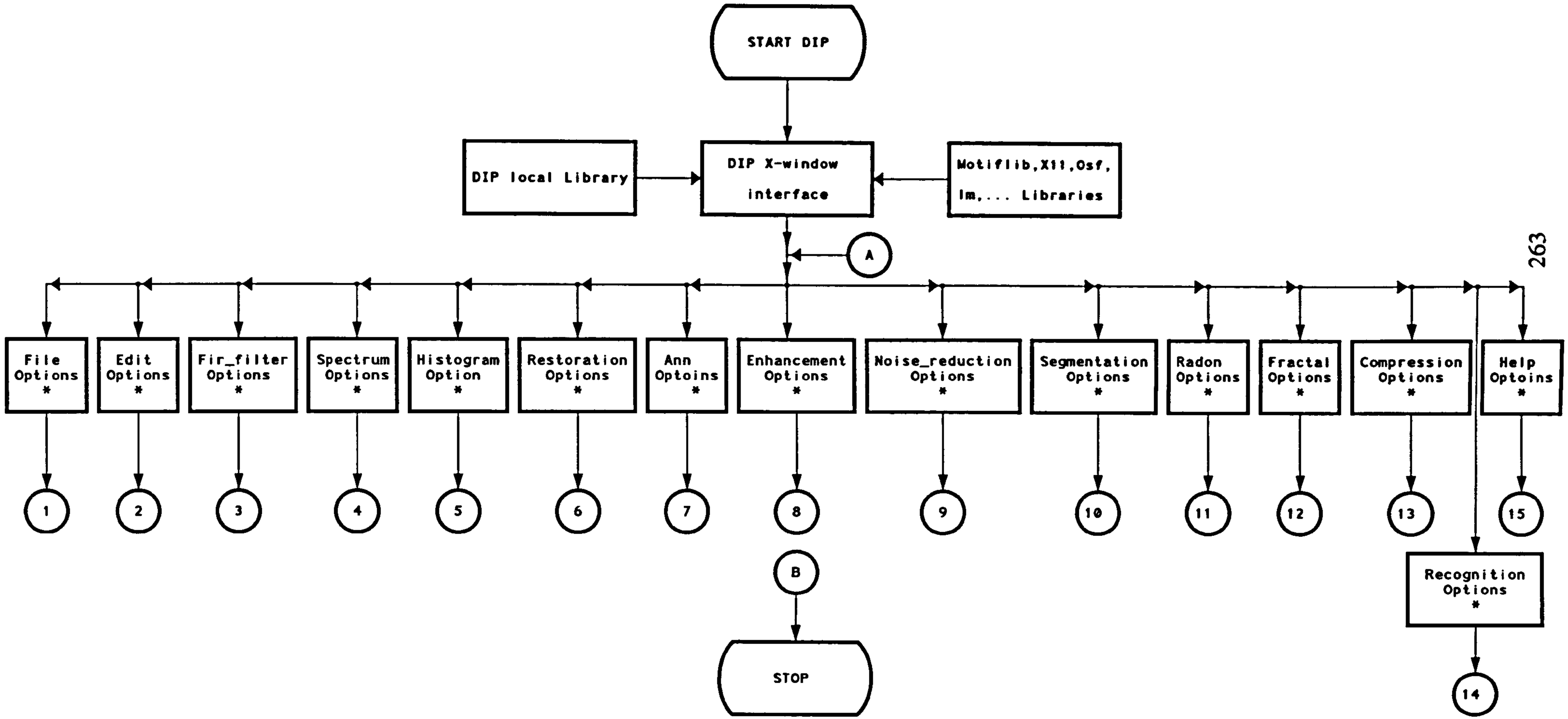
Sample Function

Function	What it does
Declaration	How it is declared
Result type	What it returns
Remarks	General information about the function

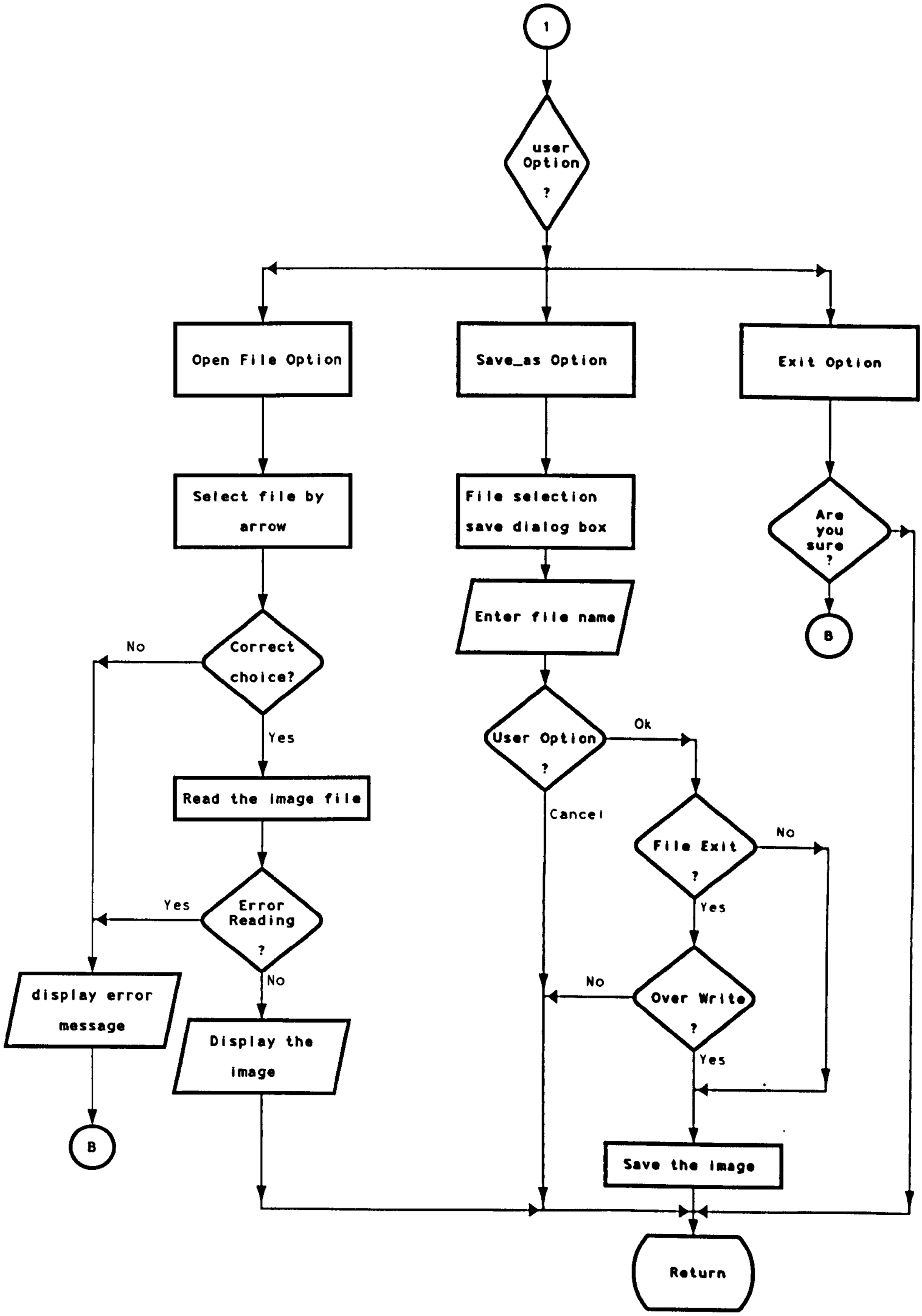
F.2. The DIP Program Flowcharts

See the following pages.

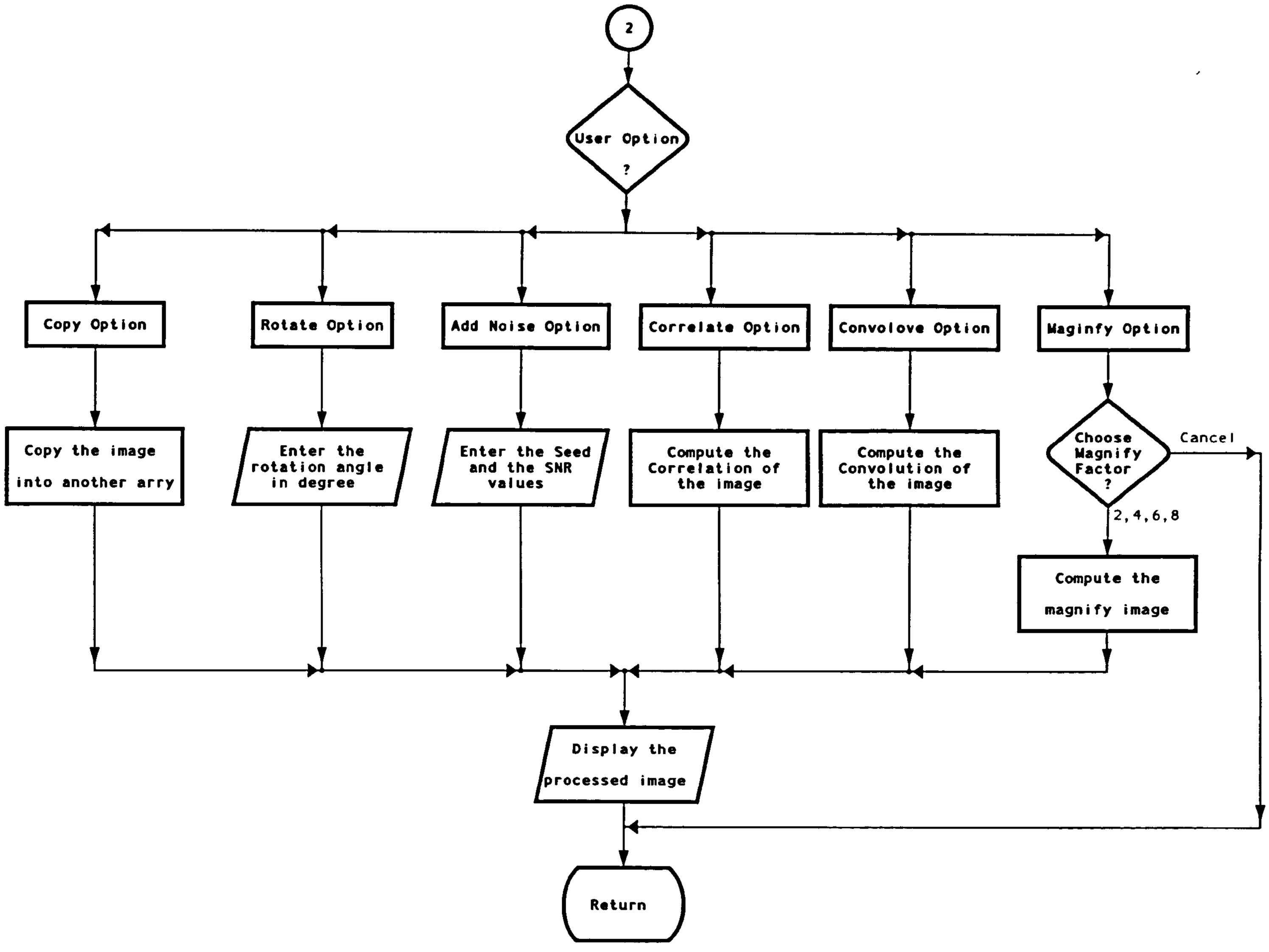
DIP Main Window Menu Flowchart



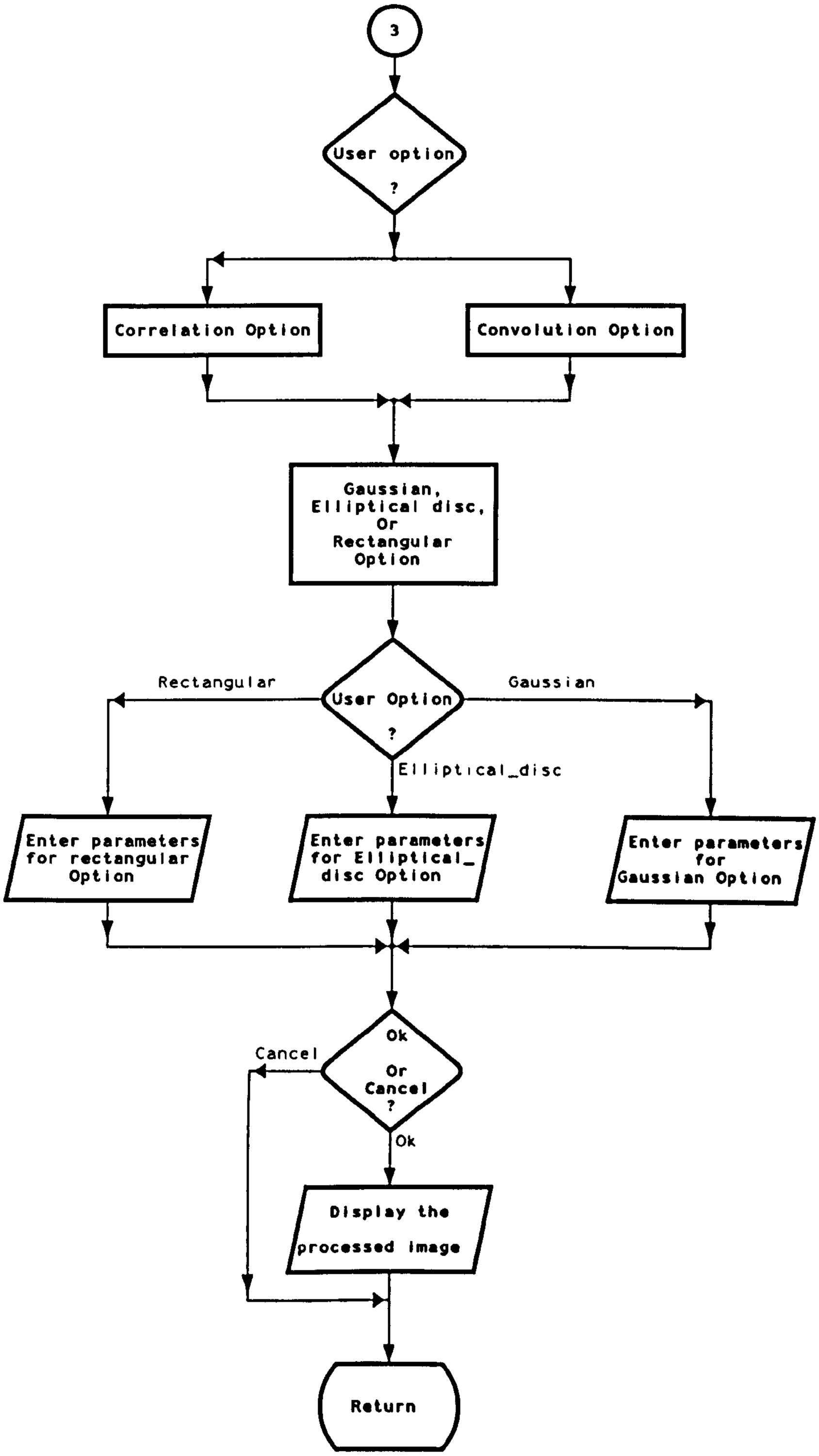
File Option Menu Flowchart



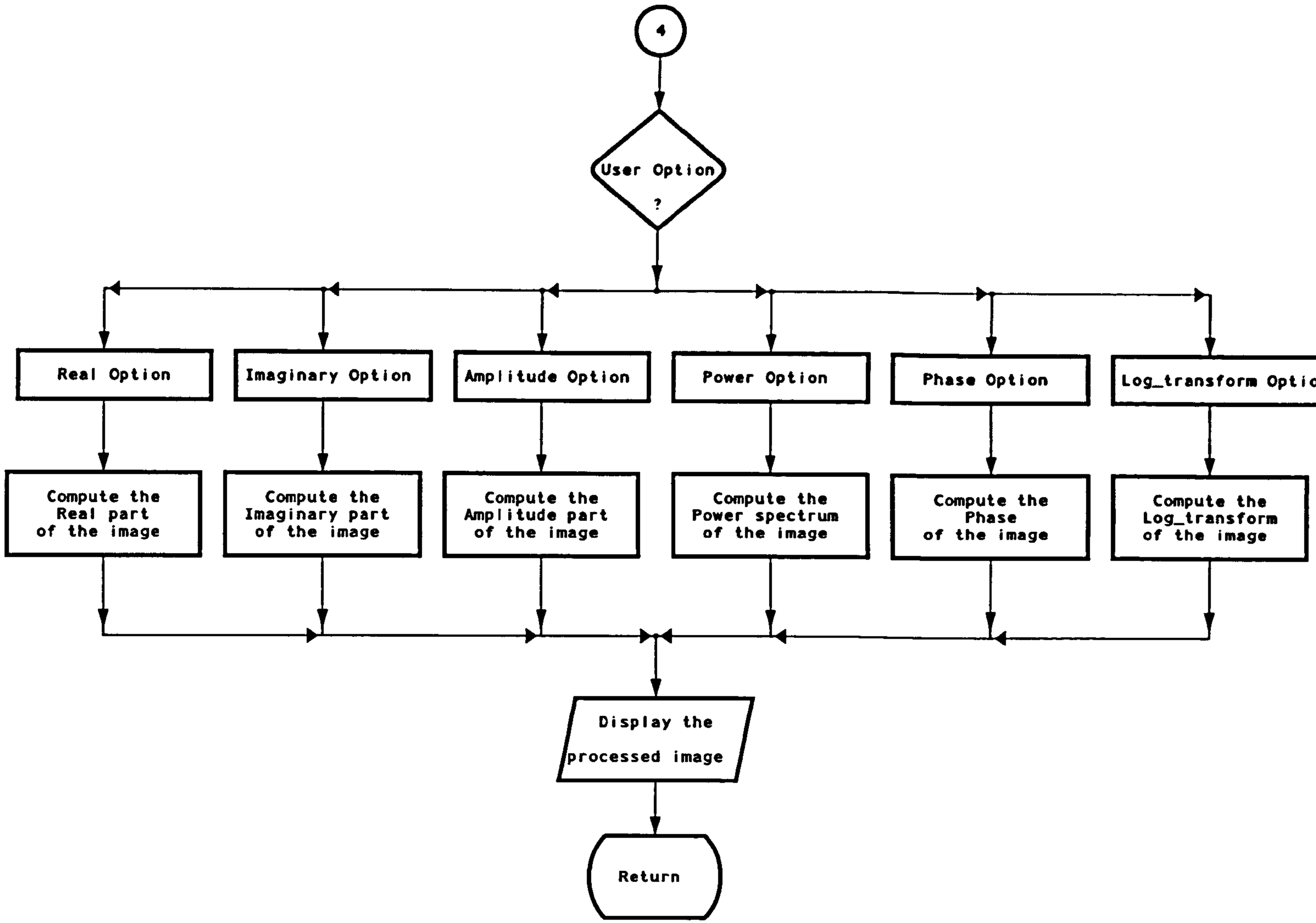
Edit Options Menu Flowchart



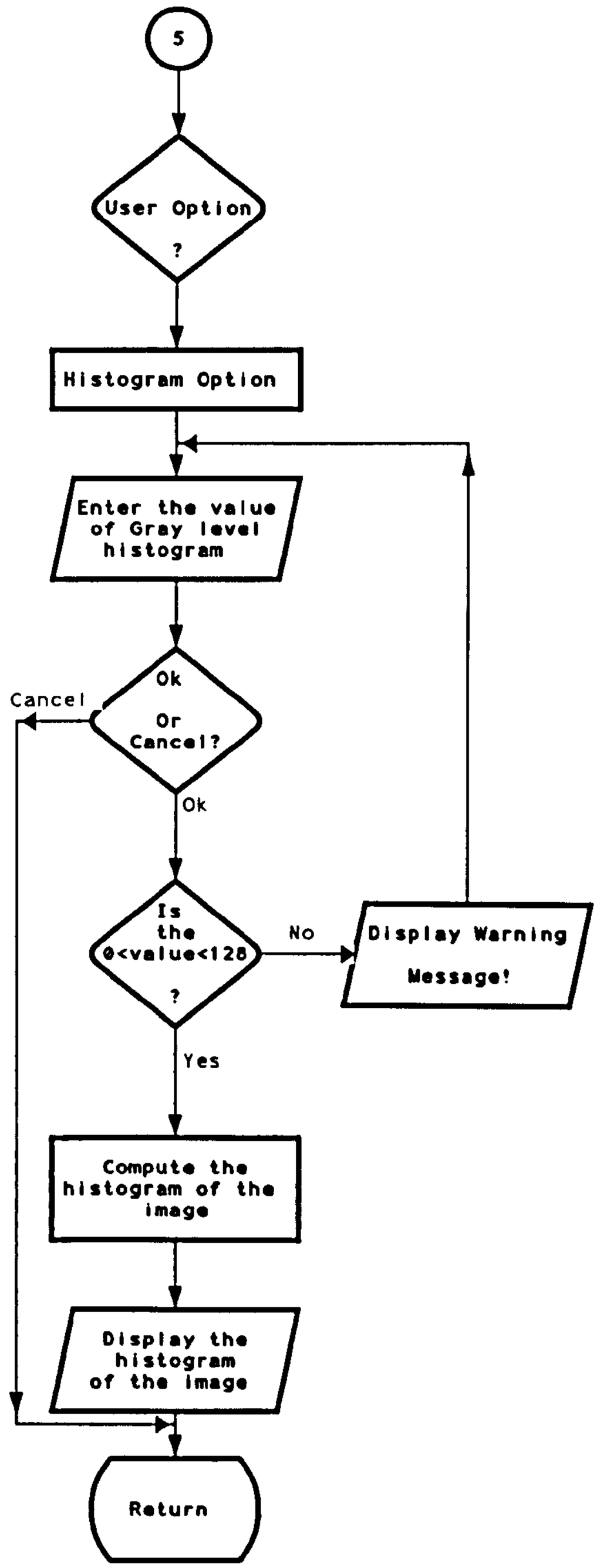
Fir_filter Option Menu Flowchart



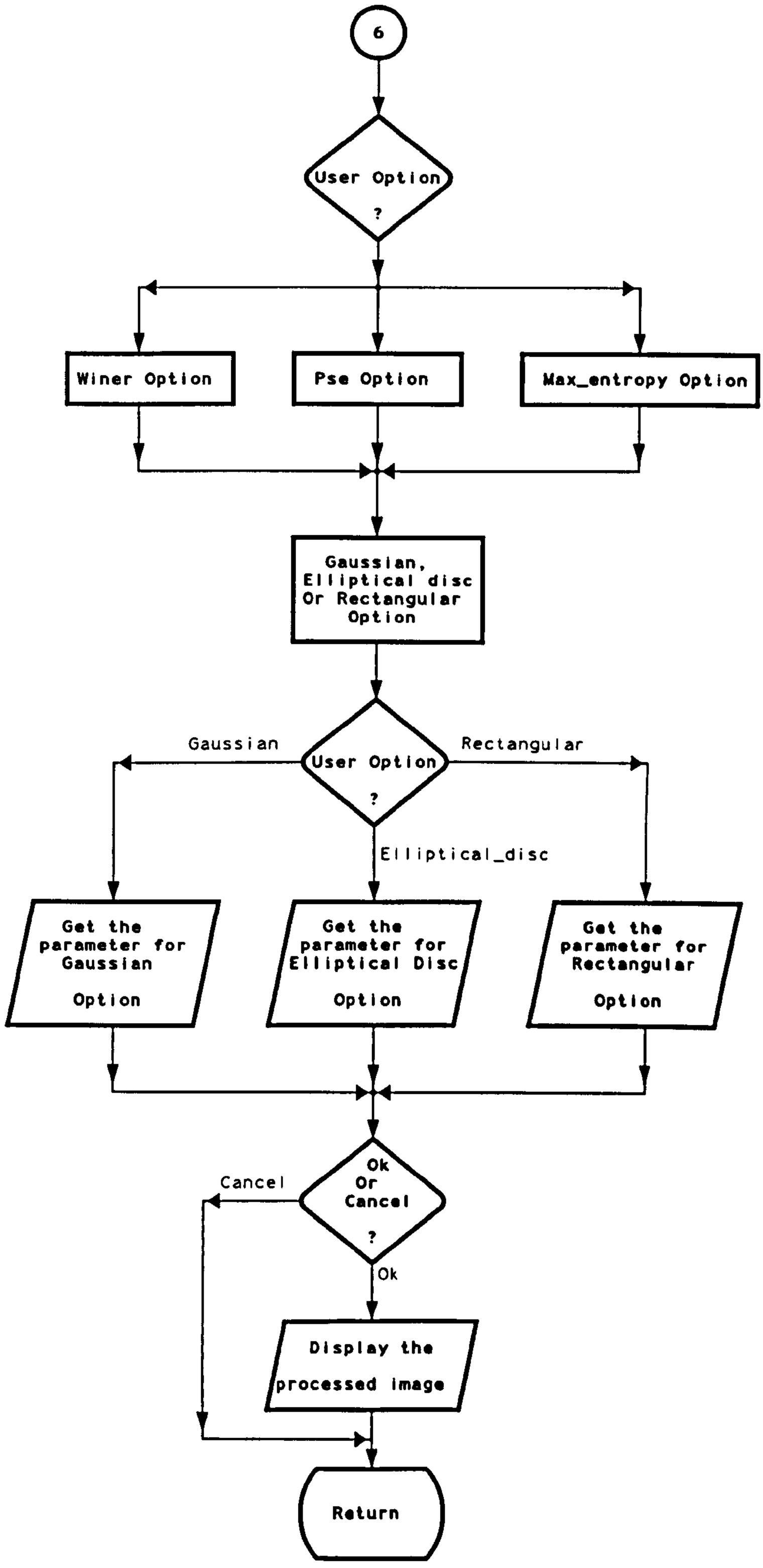
Spectrum Option Menu Flowchart



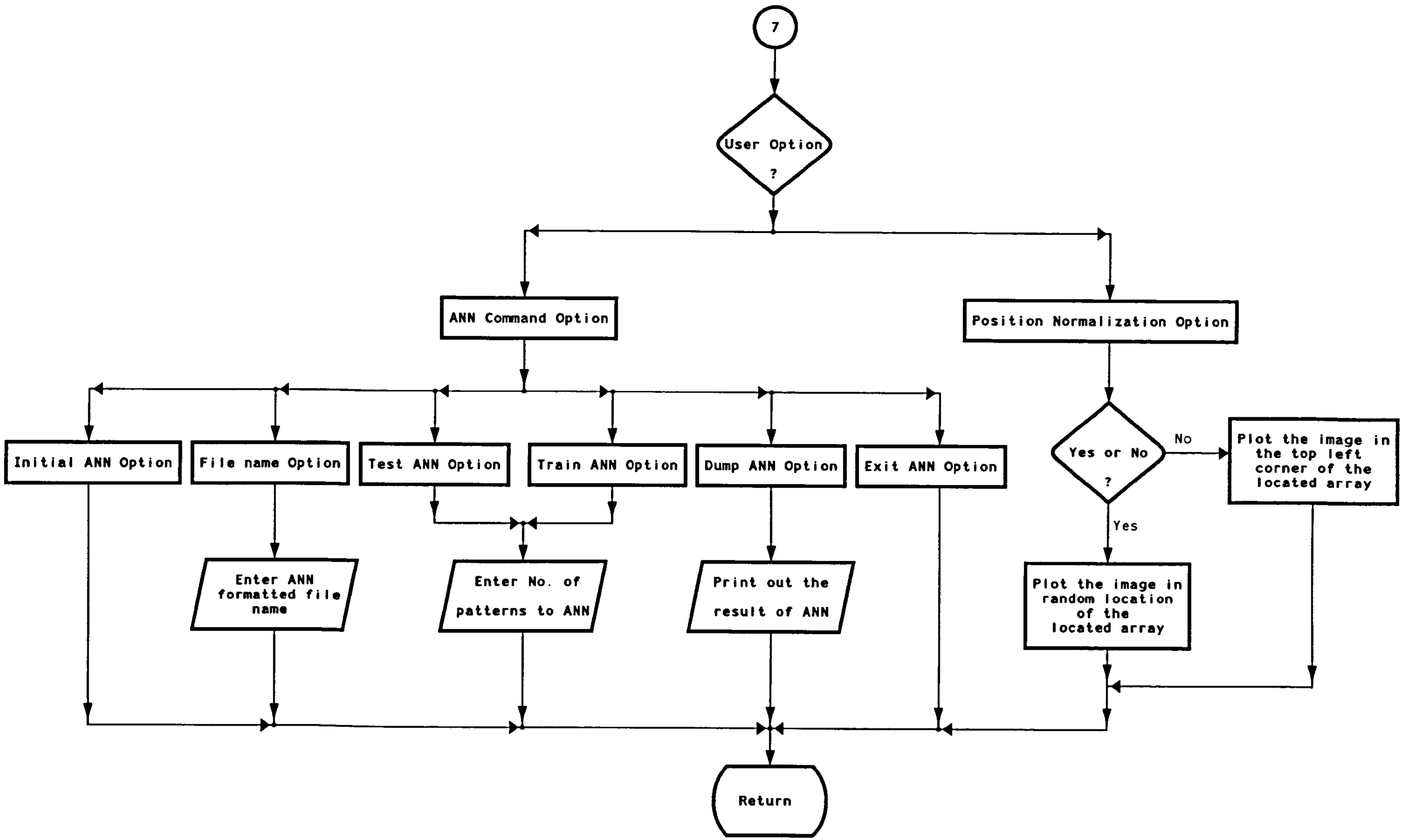
Histogram Option Menu Flowchart



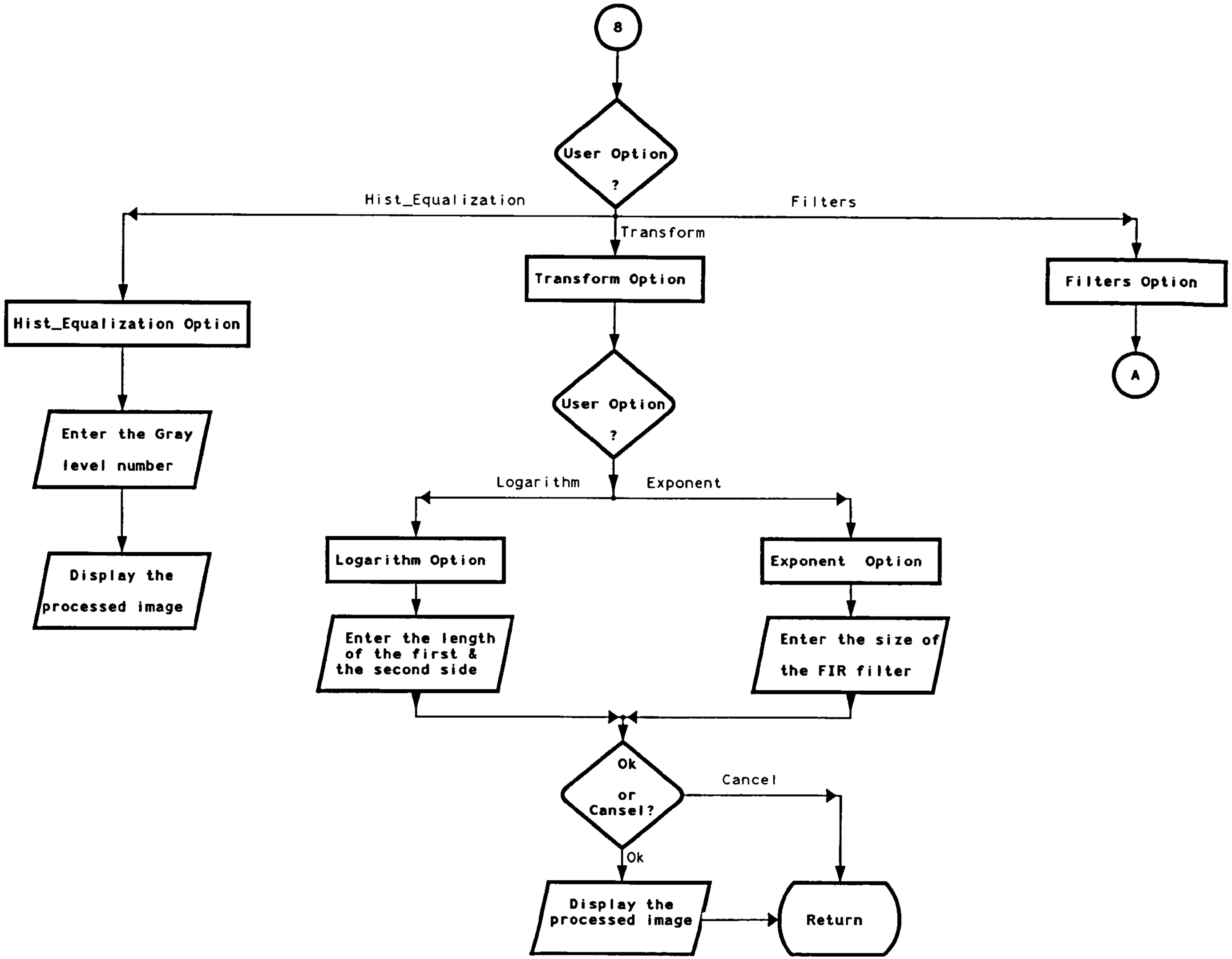
Restoration Option Menu Flowchart



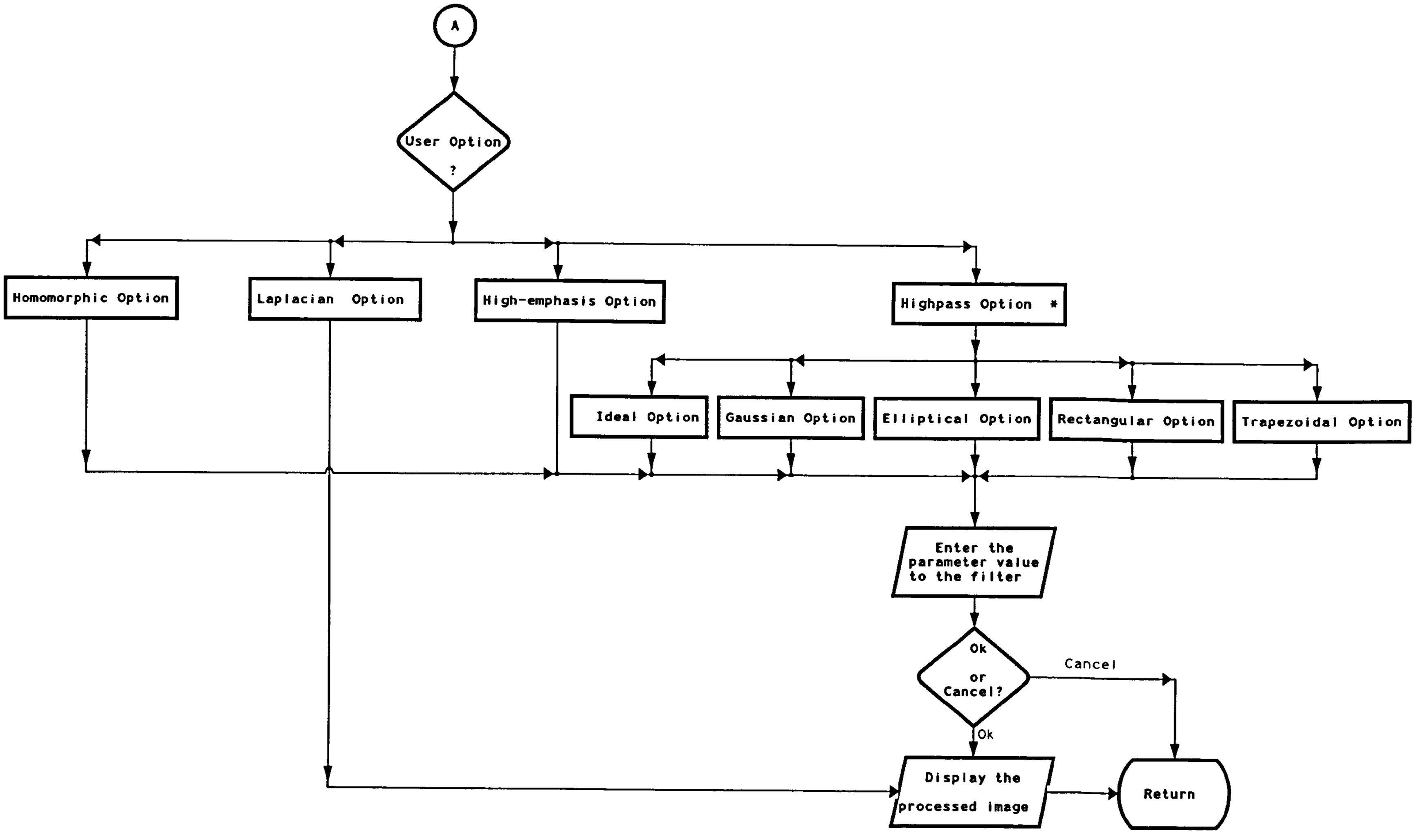
ANN Option Menu Flowchart



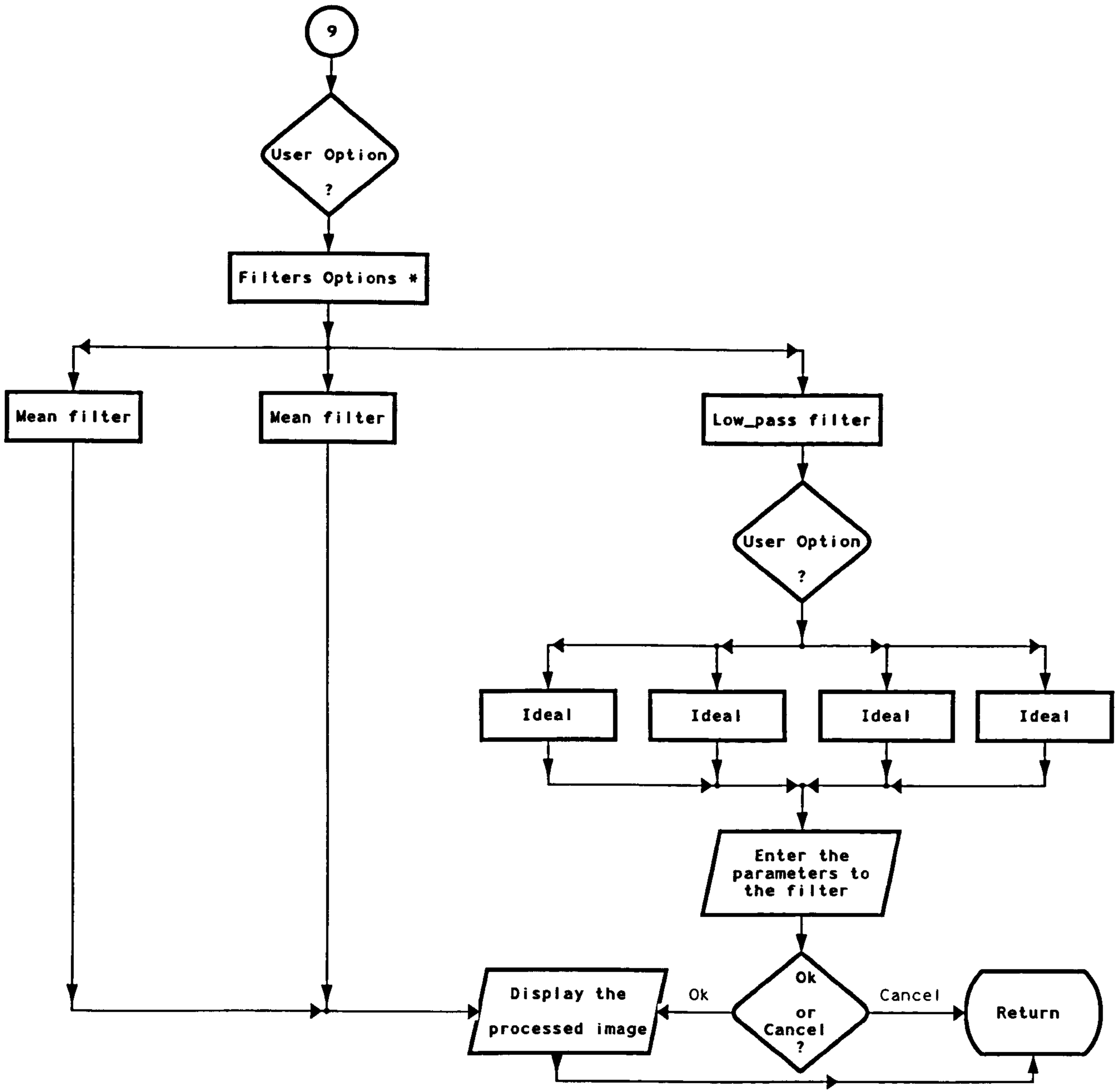
Enhancement Option Menu Flowchart



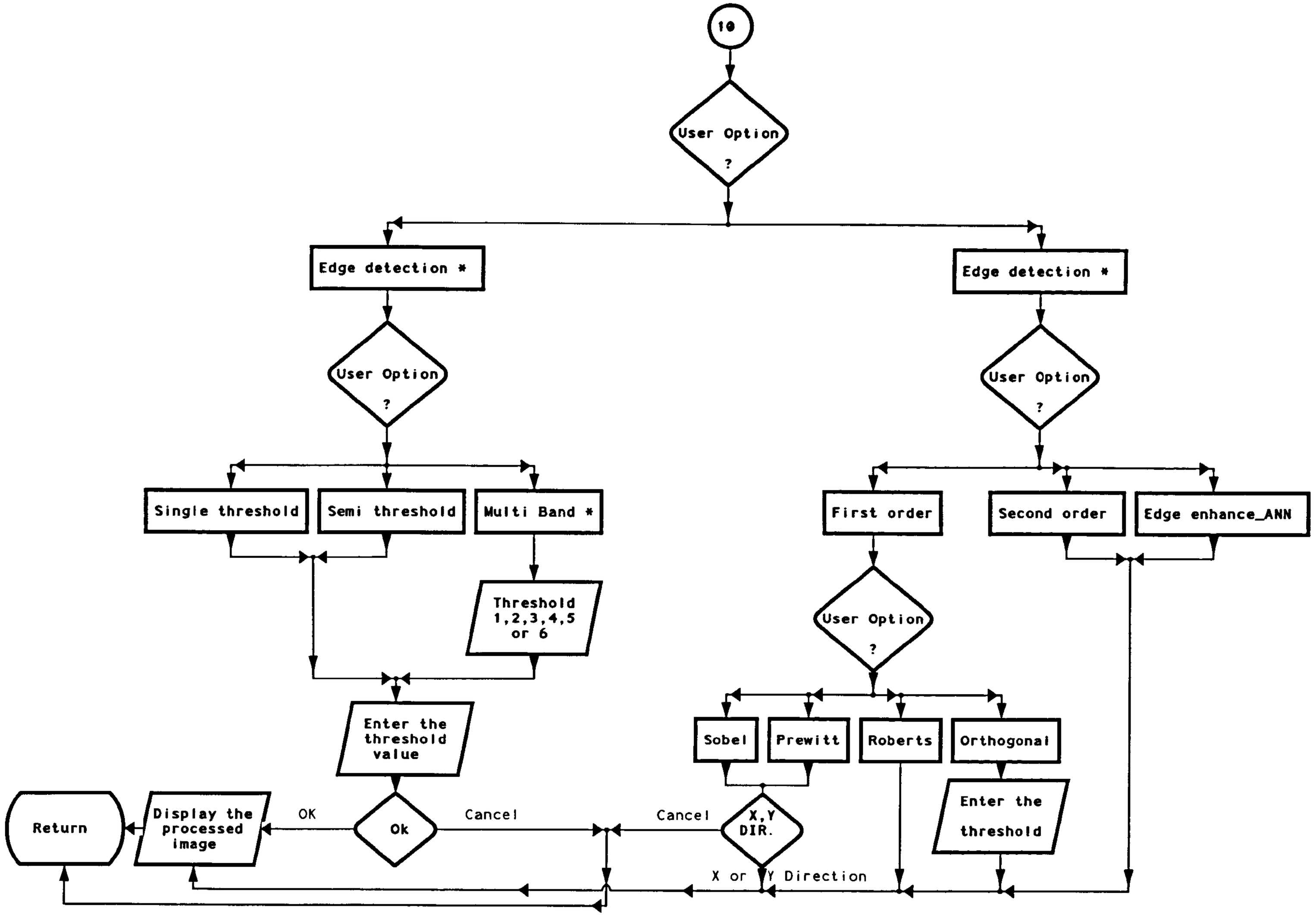
Enhancement(Continued...)



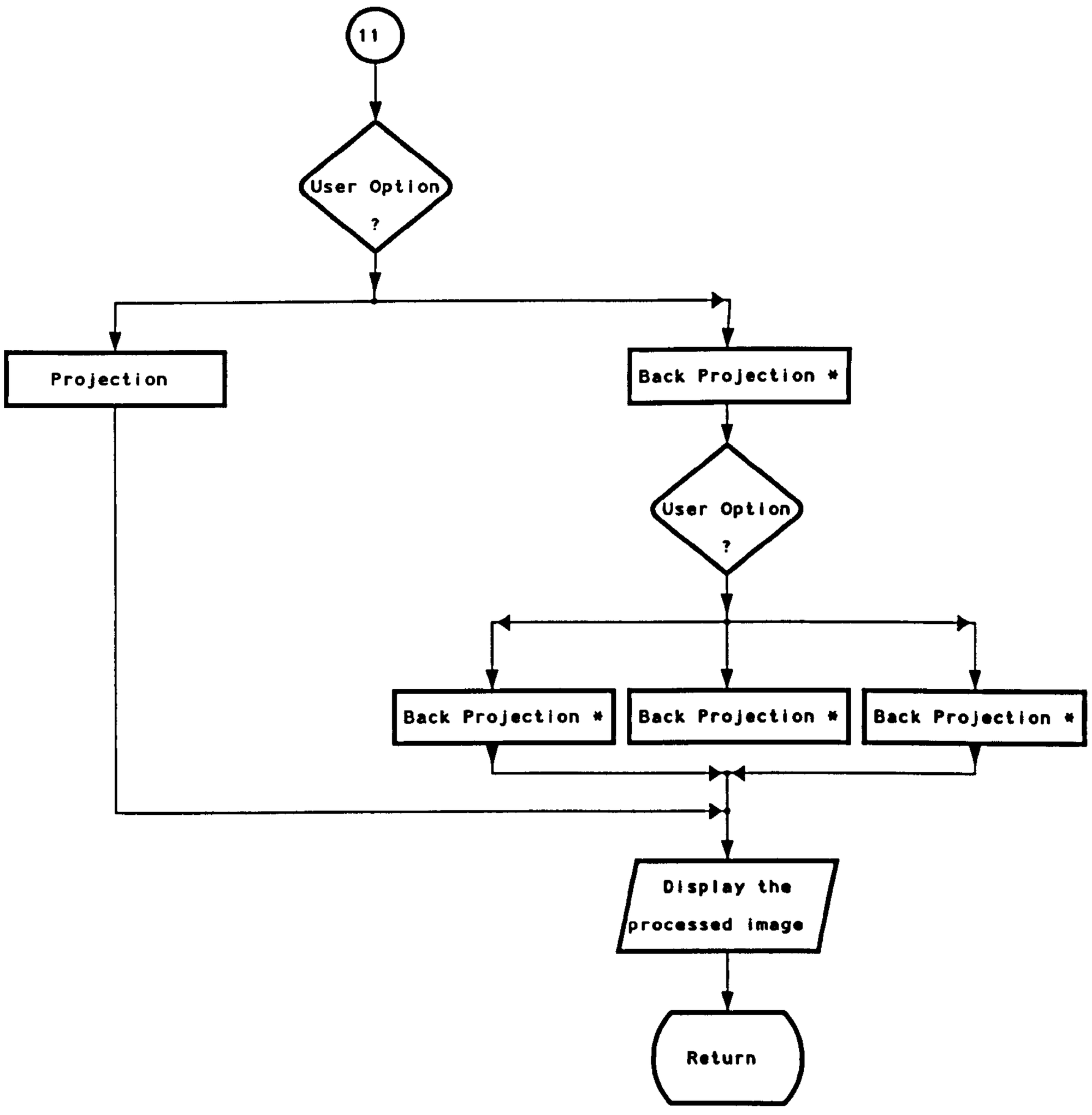
Noise reduction Option Menu Flowchart



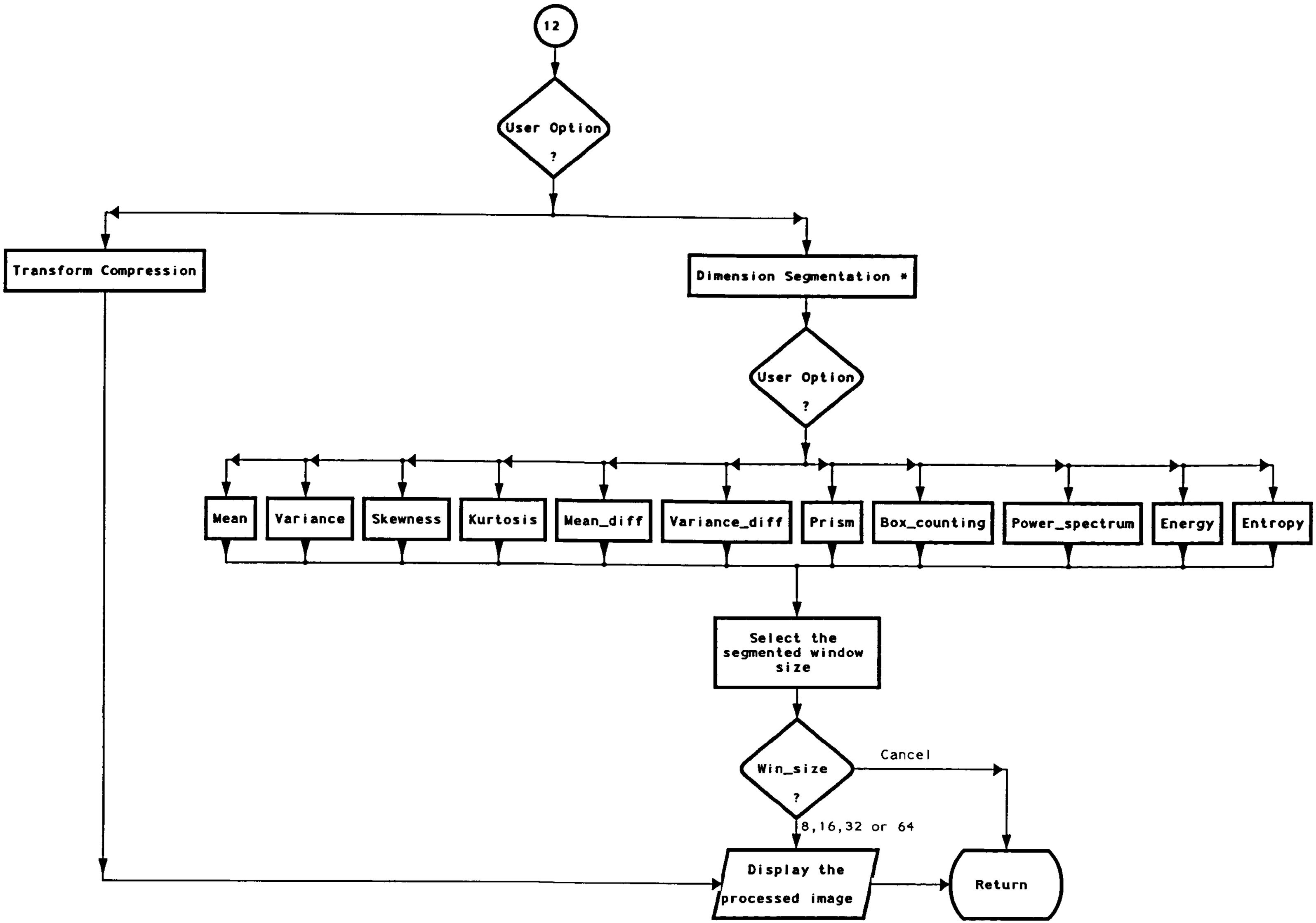
Segmentation Option Menu Flowchart



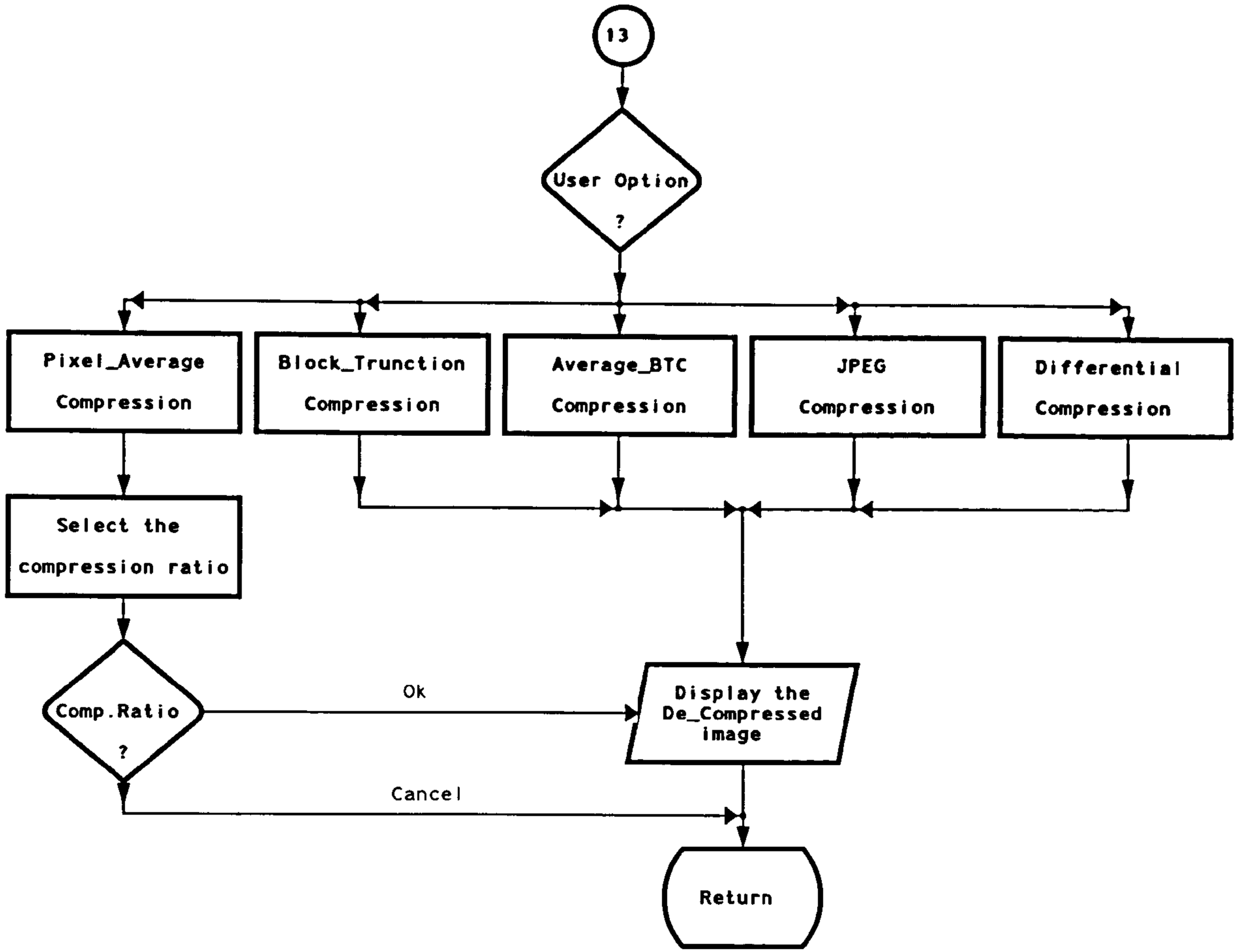
Radon Option Menu Flowchart



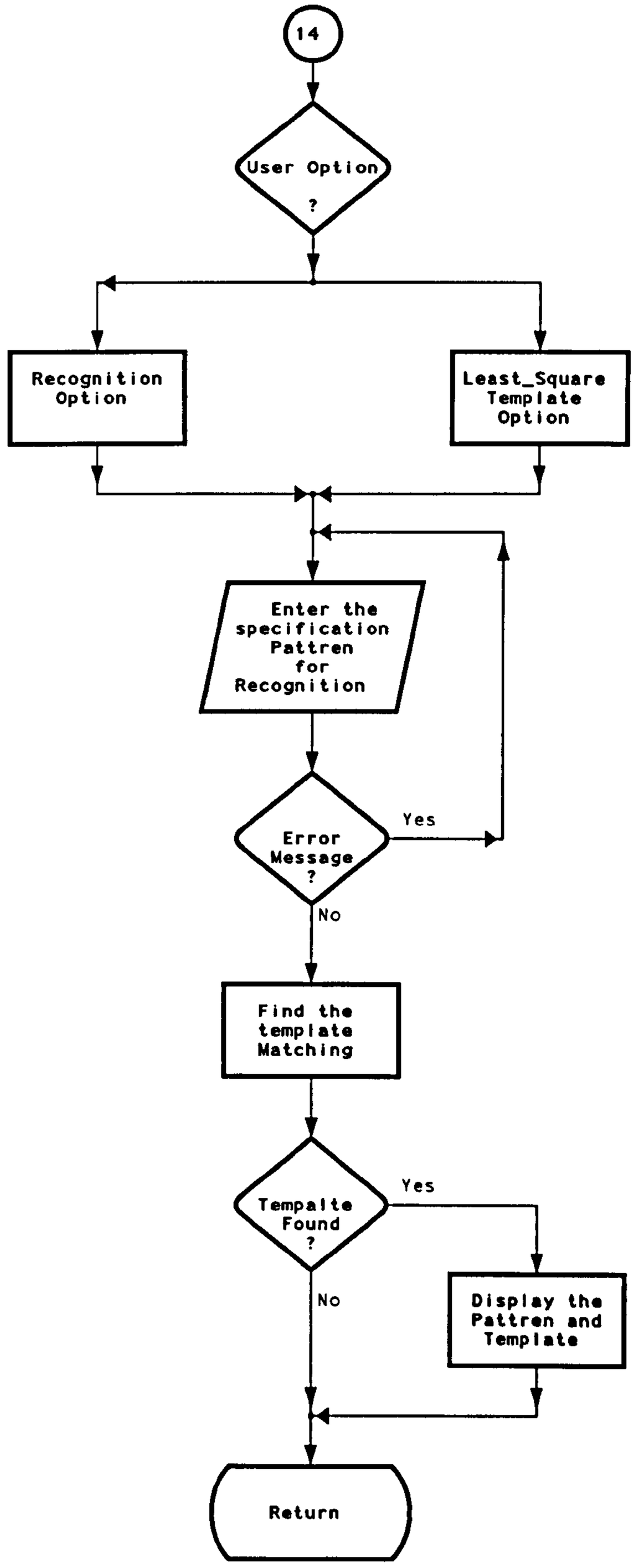
Fractal Option Menu Flowchart



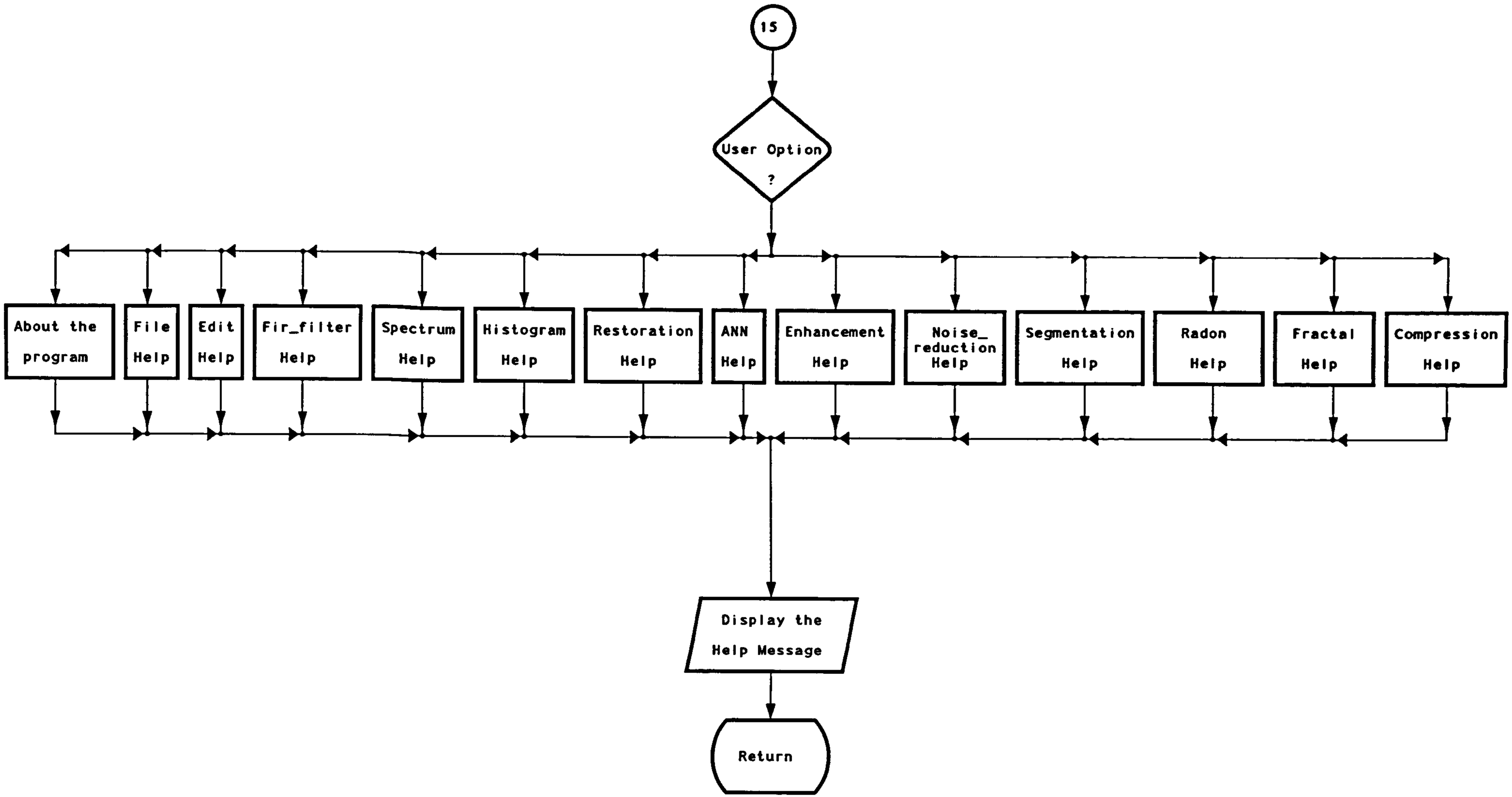
Compression Option Menu Flowchart



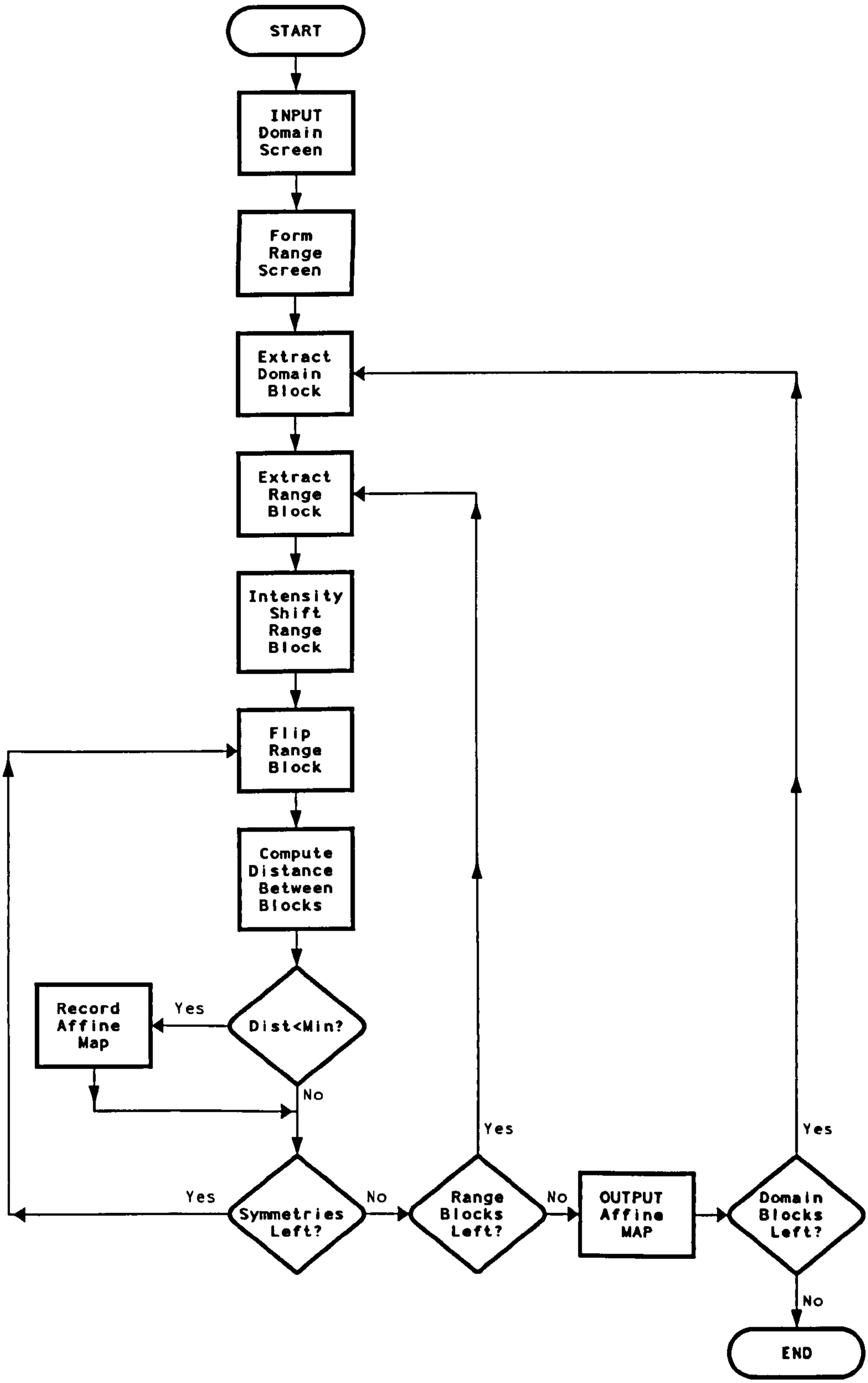
Recognition Option Menu Flowchart



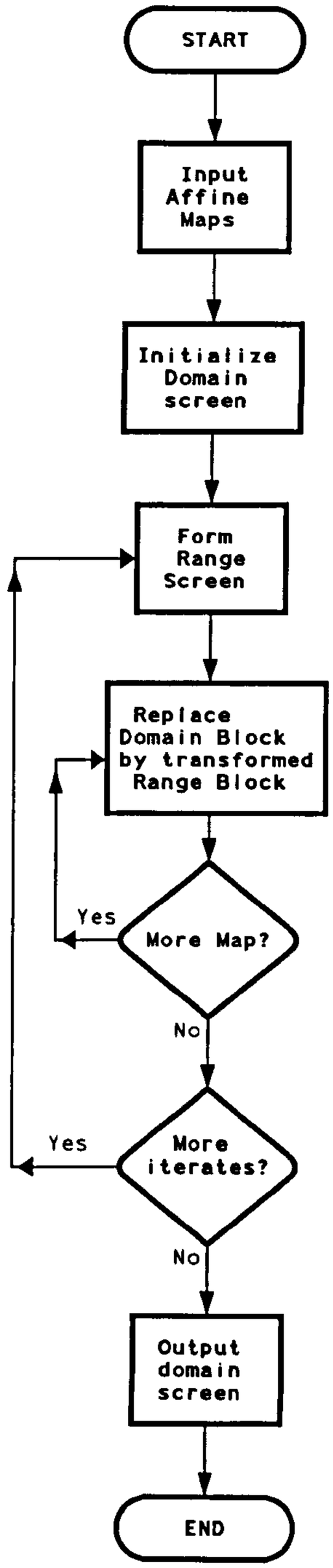
Help Option Menu Flowchart



Flowchart for the fractal transform compression



Flowchart for Decompression of the fractal transform code



F.3. The DIP algorithm function

add image	add image.c

Function Adds two square images in a ratio defined by the user.

Declaration void addimage (float **image1, float **image2,
float **sum_image, int image_size, int fsr)

Result type Output sum_image

Remarks Inputs image1, image2, image_size, fsr

External functions: float_matrix_alloc, float_matrix_dalloc

$$fsr = \frac{image1}{image2}$$

ampspec	ampspec.c

Function Computes the amplitude spectrum of an image.

Declaration void ampspec (float **image, float **ampspec, int image_size)

Result type Output the amplitude spectrum of the image

Remarks Inputs image, image_size

External functions: float_matrix_alloc, float_matrix_dalloc,
fft2d

autocor	autocor.c

Function Correlates an image with itself.

Declaration void autocor (float **image, float **auto_cor_image,
int image_size)

Result type Output auto_cor_image

Remarks Inputs image, image_size

External functions: float_matrix_alloc, float_matrix_dalloc,
fft2d

backproject	backproj.c

Function Backprojects the Radon t ranform of an image in order to reconstruct the image.

Declaration void backproject (int radon_image_size, float **radon_image, int recon_image_size, float **recon_image)

Result type Output recon_image (Reconstructed image)

Remarks Inputs radon_image_size, radon_image, recon_image_size

External functions: float_matrix_alloc, float_matrix_dalloc

bhf	bhf.c

Function Filters an image using highpass Butterworth filter.

Declaration void bhf (float **image, float **filtered_image, int image_size, int icut, int iord)

Result type Output filtered_image

Remarks Inputs image, image_size, icut (cut-off frequency), iord (order of the filter)

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

blf	blf.c

Function Filters an image using lowpass Butterworth filter.

Declaration void blf (float **image, float **filtered_image, int icut, int iord)

Result type Output filtered_image

Remarks Inputs image, icut (cut-off frequency), iord (order of the filter)

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

boxfunction	box.c

Function Routine that calculates the number of boxes of size 'size that would fit under the surface bounded by a, b, c and d.

Declaration float BOX_FUNCTION (float a, float b, float c, float d, float size)

Result type Output e (the average of the boxes)

Remarks Inputs a, b, c, d and size

btc	btc.c

Function Compresses an image using Block-Trunction-Compression algorithm.

Declaration void Block_Trunction_Comprs (float **image, int size)

Result type Output comprs.dat (image compressed file)

Remarks Inputs image, size (image size)
Internal Function: void btc (unsigned char tmp[4][4], unsigned char *a, unsigned char *b, unsigned char *mean)
to compute compressed block of size 4x4

btcd	btcd.c

Function Decompresses a compressed image using Decode Block-Trunction-Compression algorithm.

Declaration void btcd (float **Decompr_image, int image_size)

Result type Output Decompressed image file

Remarks Inputs Decompressed_image, image_size

central slice	censlice.c

Function Reconstructs an image from its Radon transform using the Central Slice theorem principal.

Declaration void centralslice (int radon_image_size, float **radon_image, int recon_image_size, float **recon_image)

Result type Output recon_image (Reconstructed image)

Remarks Inputs radon_image_size, radon_image (Radon_transformed image), recon_image_size

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

convolve	convolve.c

Function Convolves an image with a point spread function (psf).

Declaration void convolve (float **image, float **psf, float ** convolved_image, int image_size)

Result type Output convolved_image

Remarks Inputs image, image_size, psf

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

The psf has the same size as the image.

croscor	croscor.c

Function Correlates an image with a point spread function (psf).

Declaration void croscor (float **image, float **psf, float ** correlated_image, int image_size)

Result type Output correlated_image

Remarks Inputs image, image_size, psf

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

The psf has the same size as the image.

dca	dca.c

Function Compresses an image using Differential Compression algorithm.

Declaration void dca (float **image, int image_size)

Result type Output Cimage.dat (Compressed image file)

Remarks Inputs image, image_size

deconvolve	deconvol.c

Function Deconvolves a reconstruction of an obtained by straight backprojection (un-filtered) by applying the inverse filter $|K|$.

Declaration void deconvolve (float **recon_image, int recon_image_size)

Result type Output deconvolved_image

Remarks Inputs recon_image, recon_image_size

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

The input image, recon_image is affected and it is different when the function returns.

diffhilb	diffhilb.c

Function Filters a Radon transformed image by applying the filter $|K|$ in Fourier space. This operation is equivalent with the differentiation and the Hilbert transform of the image in Real

space. The image which is filtered in this way is backprojected afterwards in order to obtain a reconstruction of the original image. This method of image reconstruction is known as Filtered Back-Projection.

Declaration void diffhilb (int radon_image_size, float **radon_image, int recon_image_size, float **recon_image)

Result type Output reconstructed_image

Remarks Inputs radon_image _size, radon_image_reconstructed, image_size

External functions: float_matrix_alloc, float_matrix_dalloc, fft1d

drawhist	drawhist.c

Function Draws a bar line histogram of the data stored in histogram, with different grey levels.

Declaration void drawhist (int histogram_size, float *histogram, int graph_size, float **graph)

Result type Output graphic image (histogram image)

Remarks Inputs histogram_size, histogram, graph_size

External functions: float_matrix_alloc, float_matrix_dalloc

edgediff	edgediff.c

Function Detect the edges of an image using orthogonal gradient.

Declaration void edgediff (float **dete_edge_image, int image_size, int threshold)

Result type Output dete-edge_image

Remarks Inputs image, image_size, threshold

ehpf	ehpf.c

Function Filters an image using Highpass Exponential Filter.

Declaration void ehpf (float **image, float **filtered_image,
int image_size, int icut)

Result type Output filtered_image

Remarks Inputs image, image_size, icut(cut-off frequency)
External functions: float_matrix_alloc, float_matrix_dalloc

ellipse	ellipse.c

Function Computes an elliptical disk with given coordinates, major and minor axis and grey level.

Declaration void ellipse (float **ellipsis, int center_x, int center_y,
int major_axis, int minor_axis, int grey level)

Result type Output ellipsis

Remarks Inputs center_x, center_y, major_axis, minor_axis grey level

elpf	elpf.c

Function Filters an image using Exponential lowpass filter.

Declaration void elpf (float **image, float **filtered_image,
int image_size, int icut, int iord)

Result type Output filtered_image

Remarks Inputs image, image_size, icut(cut-off frequency),
iord (order of the filter)
External functions: float_matrix_alloc, float_matrix_dalloc
fft2d

exptran	exptran.c

Function Enhances an image by applying to it an exponential transform.

Declaration void exptran (float **image, float **enhanced_image,
int image_size)

Result type Output enhanced_image

Remarks Inputs image, image_size

fdct	fdct.c

Function Computes the forward descrite cosine transform.

Declaration void fdct(void)

Result type Output dct[8][8]

Remarks External pixels[8][8], c[8]

fft1d	fft1d.c

Function This subroutine computes the 1D DFT of a complex array whose real and imaginary parts are real-part and imaginary-part respectively, using the successive doubling method.

Declaration void fft1d (float **real_part, float **imaginary_part,
int array_size, int isgn)

Result type Output real_part, imaginary_part

Remarks Inputs real_part, imaginary_part, array_size, isgn.
The input variables real_part and imaginary_part contain the real and imaginary parts of the signal when the function is called. The function returns the real and imaginary part of the spectrum respectively. By convention the forward fourier

transform is obtained when isgn=-1 and the inverse fourier transform is obtained when isgn=1. If the input array is purely real then the imaginary part must be set to zero.

fft2d	fft2d.c

Function This subroutine computes the 2D DFT of a complex array whose real and imaginary parts are real-part and imaginary-part respectively.

Declaration void fft2d (float **real_part, float **imaginary_part, int array_size, int isgn)

Result type Output real_part, imaginary_part

Remarks Inputs real_part, imaginary_part, array_size, isgn.

External function: fft1d

The input variables real_part and imaginary_part contain the real and imaginary parts of the signal when the function is called. The function returns the real and imaginary part of the spectrum respectively. By convention the forward fourier transform is obtained when isgn=-1 and the inverse fourier transform is obtained when isgn=1. If the input array is purely real then the imaginary part must be set to zero. Zero frequency occurs at $\frac{array\_size}{2}$

fhist	fhist.c

Function Find the highest point and compute the most common pixel colour of an image.

Declaration HISTOGRAM *GENERATE_HIST (float **image,
int image_size)

Result type Output Histogram, highest point, mode

Remarks Inputs image, image_size
External functions: find_max_and_mode (Histogram * hist)

fircon	fircon.c

Function Computes the two dimensional discrete convolution (FIR) of an image with an odd square kernel. Zero padding is used.

Declaration void fircon (float **image, float **kernel,
float **convolved_image, int image_size, int kernel_size)

Result type Output convolved_image

Remarks Inputs image, kernel, image_size, kernel_size
kernel_size must be an odd integer.

fircor	fircor.c

Function Computes the two dimensional discrete correlation (FIR) of an image with an odd square kernel. For the extension of the main matrix, the zero padding technique is used.

Declaration void fircor (float **image, float **kernel,
float **correlated_image, int image_size, int kernel_size)

Result type Output correlated_image

Remarks Inputs image, image_size, kernel, kernel_size
External functions: float_matrix_alloc, float_matrix_dalloc
kernel_size must be an odd integer.

float_matrix_alloc	flmatall.c

Function Allocates memory space for a square array (2D). The memory area is initialized to zero.

Declaration float **float_matrix_alloc (int array_size)

Result type Output returns pointer to the memory area which has been allocated.

Remarks Inputs array_size

float_matrix_dalloc	flmatdal.c

Function De-allocates memory space of a square array (2D).

Declaration void float_matrix_dalloc (float **2D_array, int array_size)

Result type Output frees the allocated memory space

Remarks Inputs 2D_array (image), array_size

flt	flt.c

Function Segments an image in fixed level using the singleband thresholding method.

Declaration void flt (float **image, float **segmented_image, int image_size, float threshold)

Result type Output segmented_image

Remarks Inputs image, image_size, threshold

fractal_2d	fractal_d2.c

Function Generates a 2D 'fractal' filter in the given image. This filter is used to filter an image of Gaussian Noise to obtain an image with a given fractal dimension.

Declaration void FRACTAL_2D (float ** image, int image_size, float f_dimension)

Result type Output image (filtered image)

Remarks Inputs image, image_size, f_dimension

fract_dim_musk	fract_dim_musk.c

Function Returns the fractal dimension of a square block of data using the technique defined in mode.

Declaration float FRACTAL_DIMENSION_MUSK (float ** image, int image_size, SEGMENT_MODE mode)

Result type Output dim (fractal_dimension)

Remarks Inputs image, image_size, mode

External functions: GENERATE_HIST (image, size)

TEXTURE_PRISM_BOX (image, size, &dim, mode)

gerpap	gerpap.c

Function Reconstructs a band-limited image by applying a variation of the Gerchberg-Papoulis method, modified to accommodate a generalised weighting function w(x,y) which can be used to encode as much information as is available on the spatial characteristics of the image.

Declaration void gerpap (float **band_limited_image, float **weighting_function, int band_limited_image_extension _kspace _xdirection, int band_limited_image_extension _kspace _ydirection, int band_limited_image_size, float ** reconstructed_image)

Result type Output reconstructed_image

Remarks Inputs band_limited_image, weighting_function,
band_limited_image_extension_kspace_xdirection,
band_limited_image_extension_kspace_ydirection,
band_limited_image_size
External functions: float_matrix_alloc, float_matrix_dalloc,
ilfrect

gpsf	gpsf.c

Function Computes a Gaussian point spread function (psf) or arbitrary width.

Declaration void gpsf (float ** psf, int psf_size, int width)

Result type Output psf

Remarks Inputs psf_size, width
The width must be an integer greater than zero.

hefil	hefil.c

Function Enhances an image using the high emphasis filtering technique.

Declaration void hefil (float **image, float enhanced_image, float scaling_parameter, int image_size)

Result type Output enhanced_image

Remarks Inputs image, image_size, scaling_parameter
External functions: fircon

hist	hist.c

Function Computes the histogram of the grey levels of an image.

Declaration void hist (int, image_size, float **image, int num_grey_levels, float *histogram)

Result type Output histogram

Remarks Inputs image, image_size, num_grey_levels

External functions: float_matrix_alloc, float_matrix_dalloc

histeq	histeq.c

Function Enhances an image by applying the histogram equalization method.

Declaration void histeq (float **image, float **enhanced_image,
int image_size, int grey_levels)

Result type Output enhanced_image

Remarks Inputs image, image_size, grey_levels

External functions: float_matrix_alloc, float_matrix_dalloc

homofil	homofil.c

Function Enhances an image by applying the homomorphic filtering method. For the filtering purposes a butterworth highpass filter is used.

Declaration void homofil (float **image, float **enhanced_image,
int image_size, int icut, int iord)

Result type Output enhanced_image

Remarks Inputs image, image_size, icut (cut-off frequency),
iord (order of the filter)

External functions: float_matrix_alloc, float_matrix_dalloc, bhf
(butterworth highpass filter)

idct	idct.c

Function Computes the inverse descrite cosine transform.

Declaration void idct (void)

Result type Output pixels [8][8]

Remarks Inputs none

External variables: pixels [8][8], c[8], dct[8][8]

idhf	*idhf.c*

Function Highpass filter an image using ideal filter.

Declaration void idhf (float **image, float **filtered_image,

int image_size, int icut)

Result type Output filtered_image

Remarks Inputs image, image_size, icut (cut-off frequency)

External functions: float_matrix_alloc, float_matrix_dalloc,

fft2d

idlf	idlf.c

Function Lowpass filter an image using ideal filter.

Declaration void idlf (float **image, float **filtered_image,

int image_size, int icut)

Result type Output filtered_image

Remarks Inputs image, image_size, icut (cut-off frequency)

External functions: float_matrix_alloc, float_matrix_dalloc,

fft2d

ilfellip	ilfellip.c

Function Lowpass filter an image using an elliptical window.

Declaration void ilfellip (float **image, float **filtered_image,
int window_size_x, int window_size_y, int image_size)

Result type Output filtered_image

Remarks Inputs image, window_size_x, int window_size_y, image_size
External functions: float_matrix_alloc, float_matrix_dalloc, fft2d
Both of the window_size values must be ≤ image_size

ilfrect — ilfrect.c

Function Lowpass filter an image using a rectangular window.

Declaration void ilfrect (float **image, float **filtered_image,
int window_size_x, int window_size_y, int image_size)

Result type Output filtered_image

Remarks Inputs image, window_size_x, window_size_y, image_size
External functions: float_matrix_alloc, float_matrix_dalloc, fft2d
Both of the window_size values must be ≤ image_size

imagspec — imagspec.c

Function Computes the imaginary spectrum of an image.

Declaration void imagspec (*float* **image, *float* **spectrum,
int image_size)

Result type Output spectrum

Remarks Inputs image, image_size
External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

Initialize_network	Initialize_network.c

Function Initialize the Neural Network, the weight, the units.

Declaration void initialize_network (struct Net *netPtr)

Result type Output set the weights and the units to zero

Remarks Inputs netPtr (Net *) The Neural Network

Internal functions: randomize_weights (int num_weights, double *weight) (set all weights random), zero_unit (Unit *unitPtr) (set all Neural Network units to zero).

lapf	lapf.c

Function Outlining edges in an image using Laplacian operator.

Declaration void lapf (float **image, float **filtered_image, int image_size)

Result type Output filtered_image

Remarks Inputs image, image_size

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

line_fit	lin_fit.c

Function Given two arrays of points representing a line - it fits a straight line to them and returns the gradient.

Declaration float LINE_FIT (float *x_location, float *y_location, int no_of_points)

Result type Output returned beta

Remarks Inputs x_location, y_location, no_of_points

logspec	logspec.c

Function Enhance the amplitude spectrum of an image by applying a logarithmic transform.

Declaration void logspec (float **image, float **spectrum, int image_size)

Result type Output spectrum

Remarks Inputs image, image_size

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

logtran	logtran.c

Function Enhances details in the dark region of an image by applying a generic logarithmic transform.

Declaration void logtran (float **image, float **enhanced_image, int image_size, float scaling_parameter)

Result type Output enhanced_image

Remarks Inputs image, image_size, scaling_parameter

The scaling_parameter value must be different from zero.

magnify	magnify.c

Function Magnify an image by a factor defined by the user.

Declaration void magnify (int image_size, float **image, int magnify_factor, float ** magnified_image)

Result type Output magnified_image

Remarks Inputs image, image_size, magnify_factor

median	median.c

Function Filters an image for noise reduction purposes using the median filter principle.

Declaration void median (float **image, float **filtered_image, int image_size, int neighbourhood_size)

Result type Output filtered_image

Remarks Inputs image, image_size, neighbourhood_size

External functions: float_matrix_alloc, float_matrix_dalloc

The neighbourhood_size is the size of the window and must be an odd integer.

mflt	mflt.c

Function Segments an image in fixed levels using the multiband thresholding.

Declaration void mflt (float **image, float **segmented_image, int image_size, float *thresholds, int thresholds_number)

Result type Output segmented_image

Remarks Inputs image, image_size, thresholds, thresholds_number

Because the zero value iss considered a threshold the first element of the threshold array is expected to be zero.

minentr	minentr.c

Function Restore an image using the minimum entropy linearized filter.

Declaration void minentr (float **image, float **psf, float ** restored_image, int image_size, int longrange_multiplier)

Result type	Output restored_image
Remarks	Inputs image, image_size, psf (point spread function), longrange_multiplier External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

movav	movav.c

Function	Filters an image for noise reduction purposes using the moving average filter principle.
Declaration	void movav (float **image, float **filtered_image, int image_size, int neighbourhood_size)
Result type	Output filtered_image
Remarks	Inputs image, image_size, neighbourhood_size External functions: float_matrix_alloc, float_matrix_dalloc The neighbourhood_size is the size of the filter_window and must be an odd integer.

gnoisezd	noise.c

Function	Generates an nxn size array noise of Gaussian distributed noise with zero mean and unit variance.
Declaration	void gnoisezd (float **noise, int array_size, int seed)
Result type	Output noise
Remarks	Inputs array_size, seed Internal functions: gasdev (Return a Gaussian distributed deviate with zero mean and unit variance using RANG (IDUM) as the source of uniform deviates), rang (Return a uniform random number between 0.0 and 1.0. Set IDUM to any negative value to initiate or reinitiate the sequence).

Net_back_pass	Net_back_pass.c

Function Run Neural Network backward pass.

Declaration void network_backward_pass (Net *network)

Result type Output update the weights hidden → output
update the weights input → hidden

Remarks Inputs netPtr (Net *) The Neural Network

External functions: fan_Back (backpropagate for hidden units), update_weights (update the Neural Network weights), zero_Betas (initialize the hidden and output beta layers)

Globel: SigmoidDerivative (MACROS)

net_for_pass	net_for_pass.c

Function Run Neural Network forward pass.

Declaration void network_forward_pass (Net *network)

Result type Output update the input units

Remarks Inputs network (Net *)

Internal function: fan_in (backpropagate for input units)

phasspec	phasspec.c

Function Computes the phase spectrum of an image.

Declaration void phasespec (float **image, float **phase_spectrum, int image_size)

Result type Output phase_spectrum

Remarks Inputs image, image_size

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

pixelavrcomp	pixelavrcomp.c

Function Compresses an image by a factor defined by the user using pixel averaging principle.

Declaration void pixel_avr_comprs (int image_size, float **image, int comprs_factor, float **compressed_image)

Result type Output compressed_image

Remarks Inputs image, image_size, comprs_factor

POWAMPSPEC_2D	powampspec.c

Function Given an image, this function will produce either the amplitude or power spectrum in the output array.

Declaration void POWAMPSPEC_2D (float **image, float **output, int image_size, spectype spec)

Result type Output output

Remarks Inputs image, image_size, spec (POWER or AMPLITUDE)

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

powspec	powspec.c

Function Computes the power spectrum of an image.

Declaration void powspec (float **image, float **spectrum, int image_size)

Result type Output spectrum

Remarks Inputs image, image_size

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

prewit	prewit.c

Function Edge detects an image using Prewitt operator.

Declaration void prewit (float **image, float **edge_detect_image, int image_size, int choice)

Result type Output edge_detect_image

Remarks Inputs image, image_size

External functions: float_matrix_alloc, float_matrix_dalloc, fircon

Choice is the direction for edge detection which is an integer value. 1 stands for x_direction and 2 stands for y_direction.

prism_box	prism_box.c

Function To segment an image along the lines of texture. The function computes the fractal dimension of an image at each pixel by using the prism method. This basically splits the image up into a number of sub-squares. The central pixel is calculated as the mean of the four corners. The area of the 4 triangles is then calculated and the process repeated for the next square. Having covered the entire image, the square size is halved and the process repeated. If a log-log plot is constructed for square size against total area, this gives a curve that if a straight line is drawn through the curve the fractal dimension can be found from this line. For exact fractal surfaces, this curve is a straight line.

Declaration error_struct *TEXTURE_PRISM_BOX (float **image, int image_size, float *fractal_dimension, SEGMENT_MODE mode)

Result type Output fractal_dimension

Remarks Inputs image, image_size, mode (fractal dimension technique either BOX_COUNTING or PRISM)
External functions: PRISM_FUNCTION, BOX_FUNCTION, LINE_FIT

project	project.c

Function Sums over the elements which lie on a row of 2D array given in this way a 1D array.

Declaration void project (float **2d_array, int array_size, float **1d_array)

Result type Output 1d_array

Remarks Inputs 2d_array, array_size

pse	pse.c

Function Restores a blurred image using the Power Spectrum Equalization filter.

Declaration void pse (float **blurred_image, float **psf, float **restored_image, int blurred_image_size, float SNR)

Result type Output restored_image

Remarks Inputs blurred_image, psf, blurred_image_size, SNR
External functions: float_matrix_alloc, float_matrix_dalloc, fft2d
The blurred_image_size must be power of two.

radon	radon.c

Function Computes the Radon transform of an image.

Declaration void radon (float **image, int image_size, float **transformed_image, int transformed_image_size)

Result type Output transformed_image

Remarks Inputs image, image_size, transformed_image_size

External functions: float_matrix_alloc, float_matrix_dalloc, rotate, project

rdascf	rdascf.c

Function Reads an image which has been written in ascii format from a named data file.

Format type: height of the image (ascii form)
width of the image (ascii form)
image data (ascii form)

Declaration int rdascf (char *file_name, float ***image, int *image_size)

Result type Output image, image_size

Returned value: -1 if read error occurred
1 if valid image has been read

Remarks Inputs file_name

External functions: float_matrix_alloc, float_matrix_dalloc

The memory for the image pointer is allocated inside the function.

rdxvascf	rdxvascf.c

Function Reads an image which has been written in binary (ascii) format from a named data file.

Format type: P2 format identifier
width of the image (integer)
height of the image (integer)
image data (positive integer)

Declaration int rdxvascf (char *file_name, float ***image, int *image_size)

Result type Output image, image_size

Returned value: -1 if read error occurred

1 if valid image has been read

Remarks Inputs file_name

External functions: float_matrix_alloc, float_matrix_dalloc

The memory for the image pointer is allocated inside the function.

realspec	realspec.c

Function Computes the real spectrum of an image.

Declaration void realspec (float **_image, float **spectrum,

int image_size)

Result type Output spectrum

Remarks Inputs image, image_size

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

roborts	roborts.c

Function Edge detect an image using the Roborts gradient.

Declaration void roborts (float **image, float **edge_detected_image,

int image_size)

Result type Output edge_detected_image

Remarks Inputs image, image_size

rotate	rotate.c

Function Rotates an image (anti-clockwise) through a given angle using the nearest neighbour approximation (with corner clipping).

Declaration void rotate (float **image, int image_size, float **rotated_image, int rotated_image_size, float theta)

Result type Output rotated_image

Remarks Inputs image, image_size, rotated_image_size, theta (the angle of rotation defined in degrees)

secoredg	secoredg.c

Function Edge detect an image using second order edge detection.

Declaration void secoredg (float **image, float **edge_detected_image, int image_size)

Result type Output edge_detected_image

Remarks Inputs image, image_size

External functions: float_matrix_alloc, float_matrix_dalloc, fircon

sincint	sincint.c

Function Interpolate an input image from 2**n to 2**m using FFT and the principle of sinc interpolation.

Declaration void sincint (float **original_image, int original_image_size, float **interpolated_image, int interpolated_image_size)

Result type Output interpolated_image

Remarks Inputs original_image, original_image_size, interpolated_image_size

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

Interpolated_image_size must be ≥ original_image_size.

slt	slt.c

Function Segments an image in fixed level using the semi-thresholding method.

Declaration void slt (float **image, float **segmented_image, int image_size, float threshold)

Result type Output segmented_image

Remarks Inputs image, image_size, threshold

sobel	sobel.c

Function Edge detect an image using the soble operator.

Declaration void sobel (float **image, float **edge_detected_image, int image_size, int choice)

Result type Output edge_detected_image

Remarks Inputs image, image_size, choice

External functions: float_matrix_alloc, float_matrix_dalloc, fircon

Choice is the direction of the edge detection: 1 stands for x_direction and 2 stands for y_direction.

square	square.c

Function Computes a rectangular area with given coordinates, sides and grey level.

Declaration void square (float **square, int center_x, int center_y, int x_side, int y_side, int grey_level)

Result type Output square

Remarks Inputs center_x, center_y, x_side, y_side, grey_level

x_side must be ≤ 2*center_x, y_side must be ≤ 2*center_y

thpf	thpf.c

Function Filters an image for noise reduction purposes using Trapezoidal high pass filter.

Declaration void thpf (float **image, float **filtered_image, int image_size, int icut)

Result type Output filtered_image

Remarks Inputs image, image_size, icut (cut-off frequency)

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

tlpf	tlpf.c

Function Filters an image for noise reduction purposes using Trapezoidal low pass filter.

Declaration void tlpf (float **image, float **filtered_image, int image_size, int icut)

Result type Output filtered_image

Remarks Inputs image, image_size, icut (cut-off frequency)

External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

undca	undca.c

Function Decompresses an image compressed by differential_ compression_algorithm.

Declaration void undca (void)

Result type Output reconstructed original image data

Remarks Inputs compressed image_data_file

wiener	wiener.c

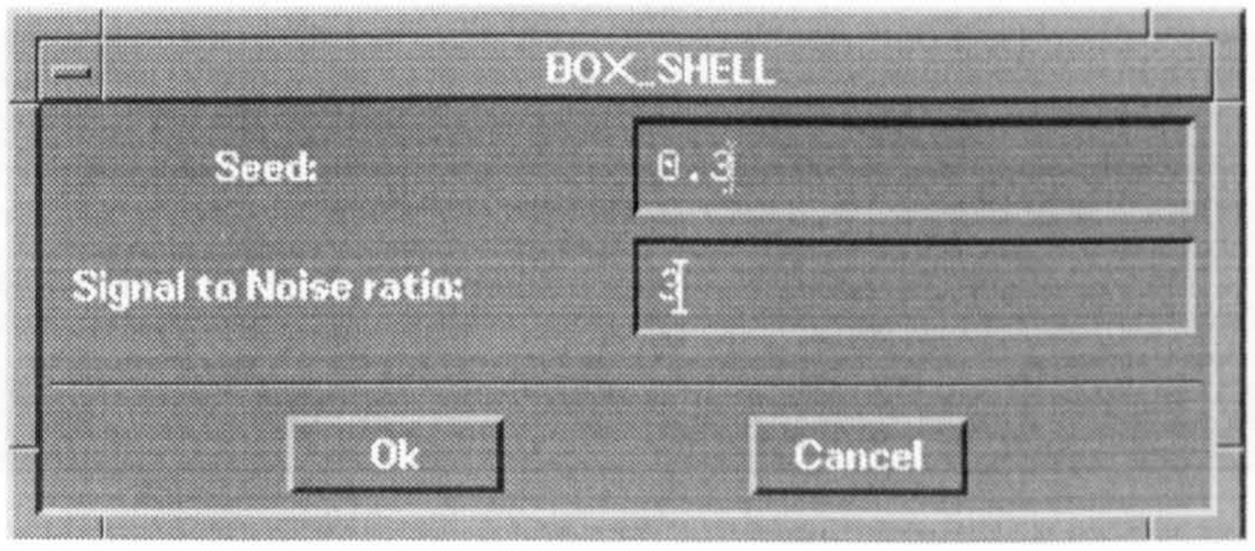
Function	Restores a blurred image using the Wiener filter.
Declaration	void wiener (float **blurred_image, float **psf, float **restored_image, int blurred_image_size, float SNR)
Result type	Output restored_image
Remarks	Inputs blurred_image, psf, blurred_image_size, SNR External functions: float_matrix_alloc, float_matrix_dalloc, fft2d

Appendix G

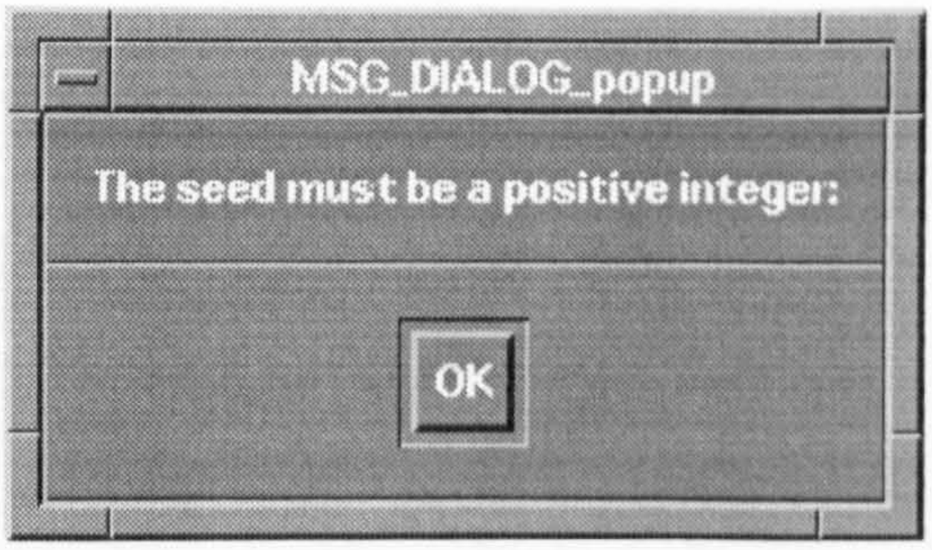
The DIP Warning Messages

This appendix displays some the warning messages that DIP detects user mistakes and responds by giving a suitable warning, depending on the type of user mistake. These warning messages will appear whenever the user types or enters wrong data to the package and the DIP program will not respond to the user request until the user corrects his/her mistake.

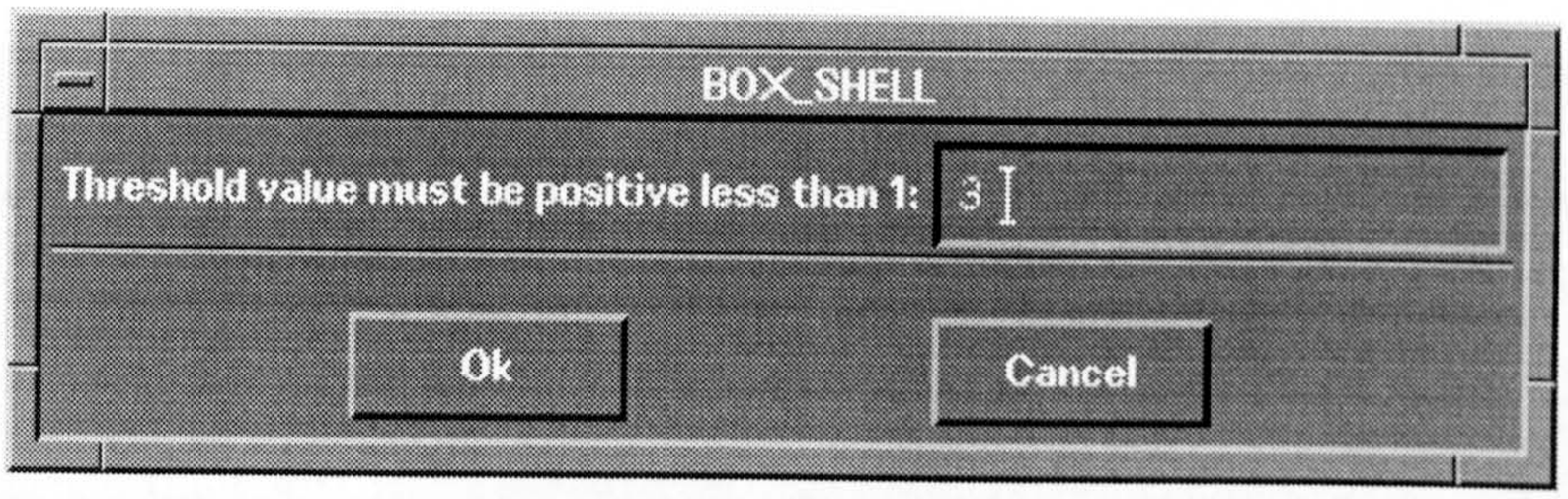
The following figures are some examples of user mistakes and the warning messages that the DIP package shows up.



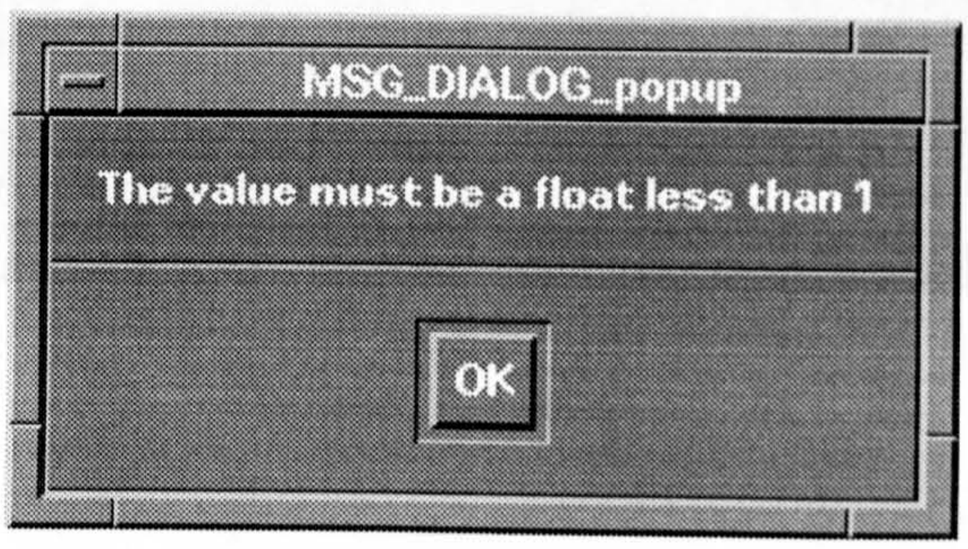
The user mistake



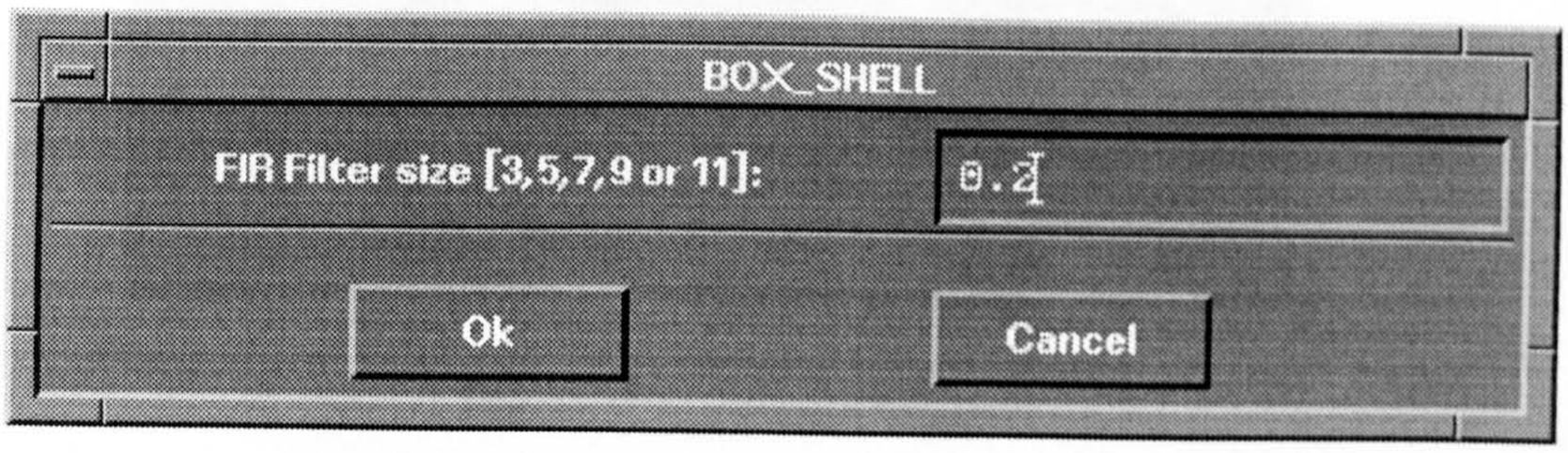
The DIP warning message



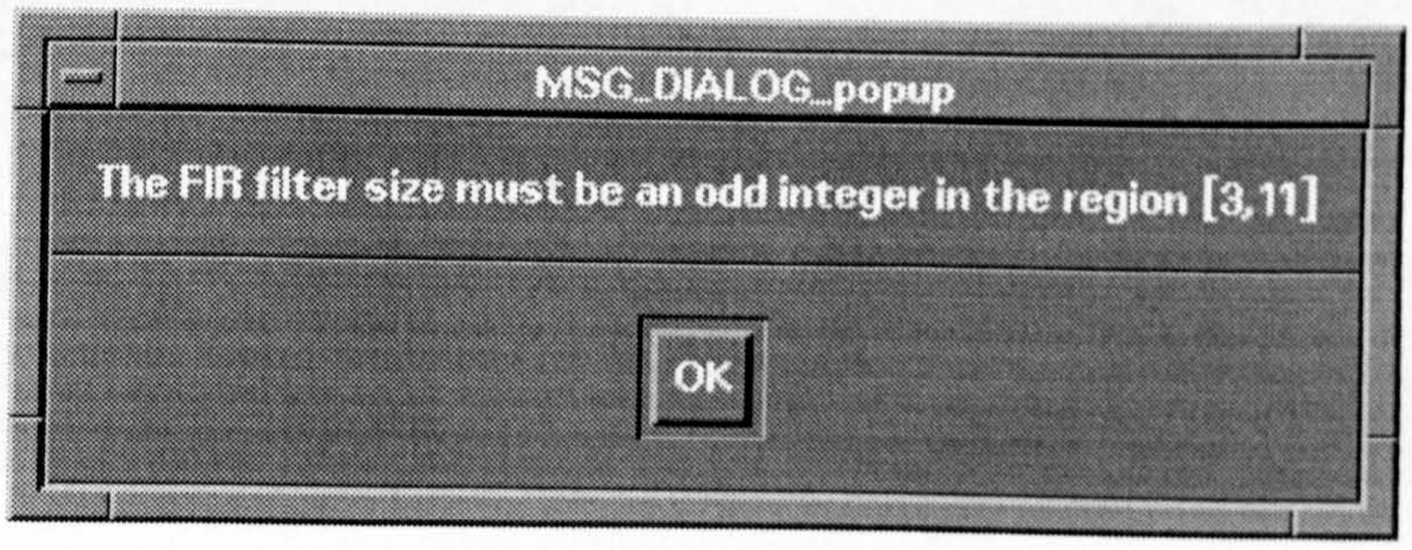
The user mistake



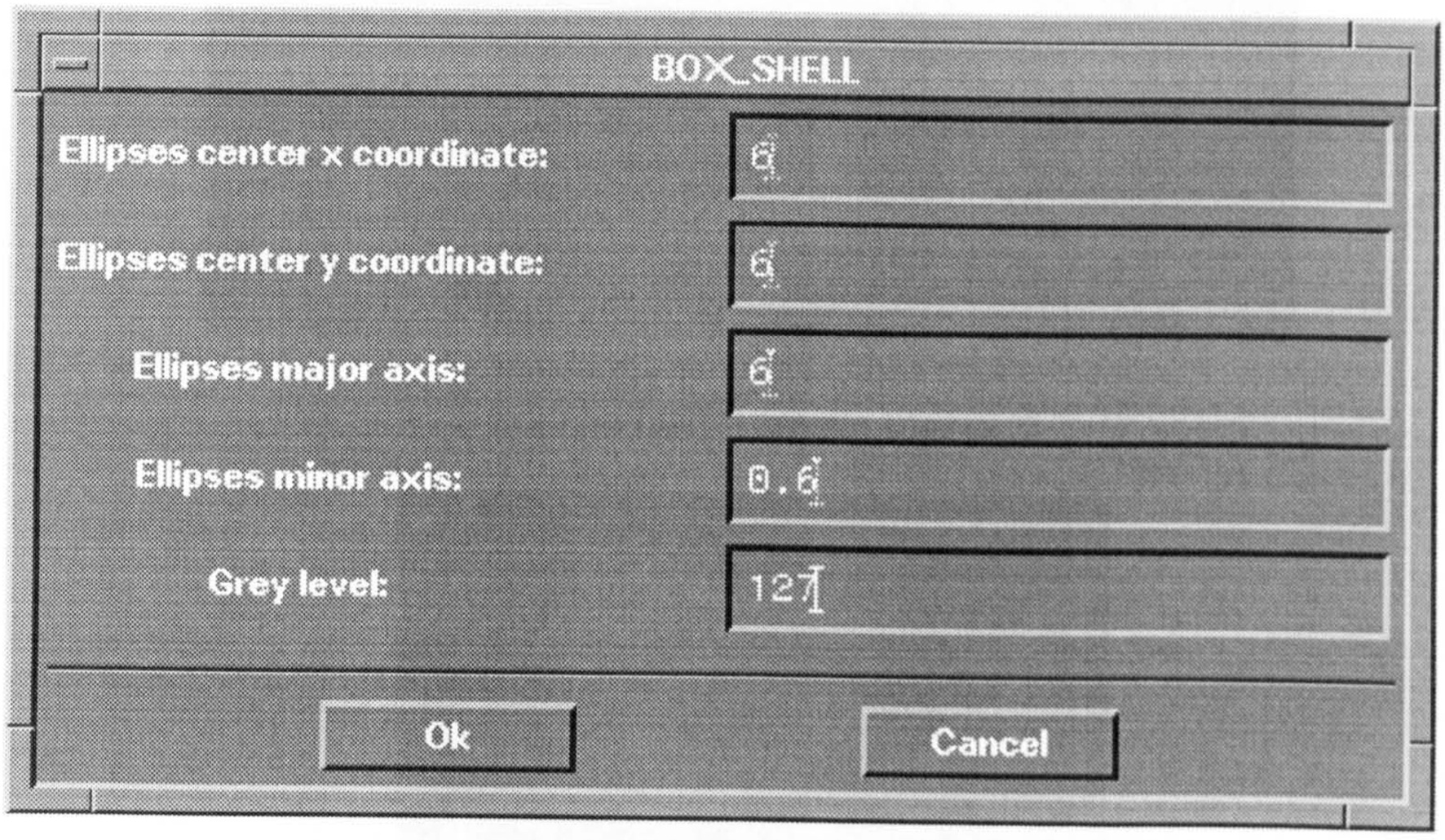
The DIP warning message



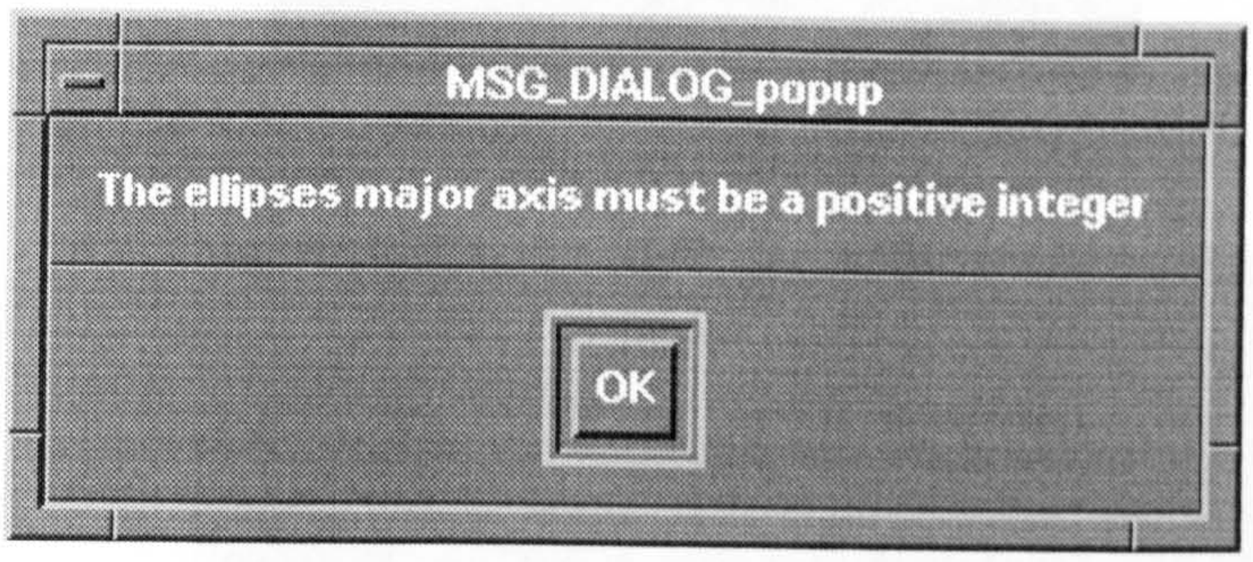
The user mistake



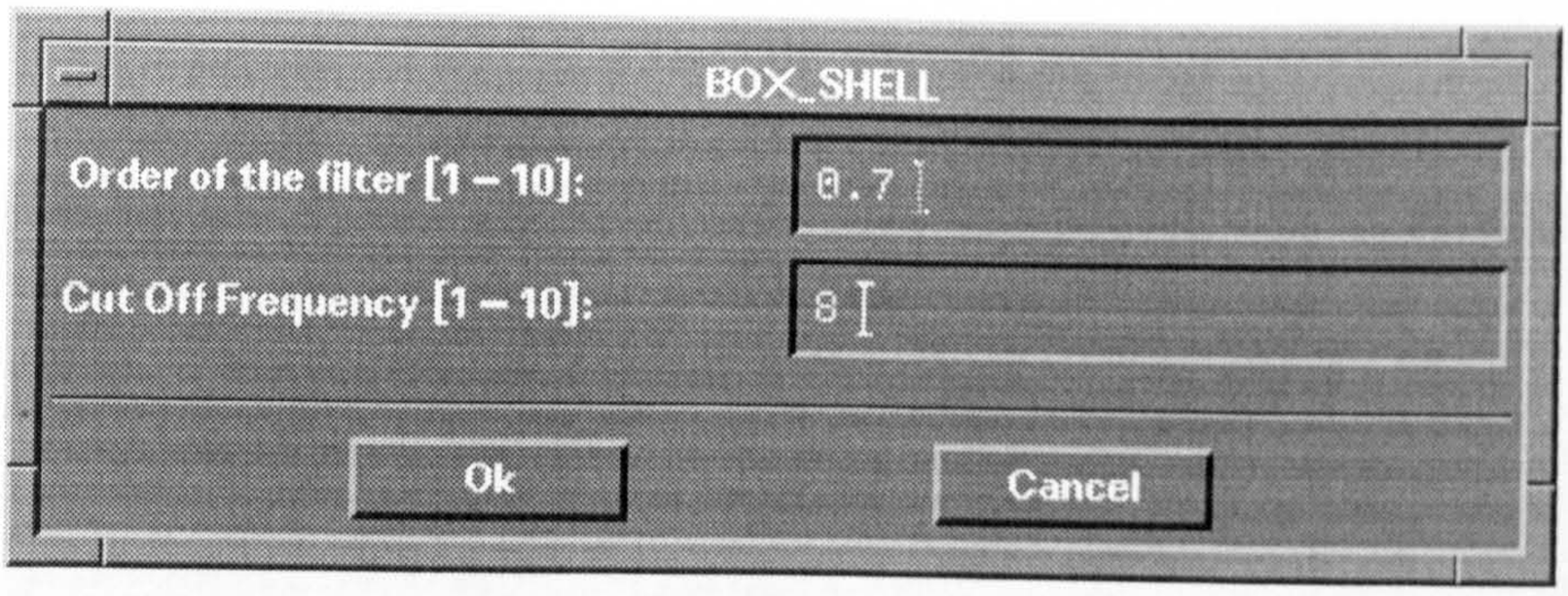
The DIP warning message



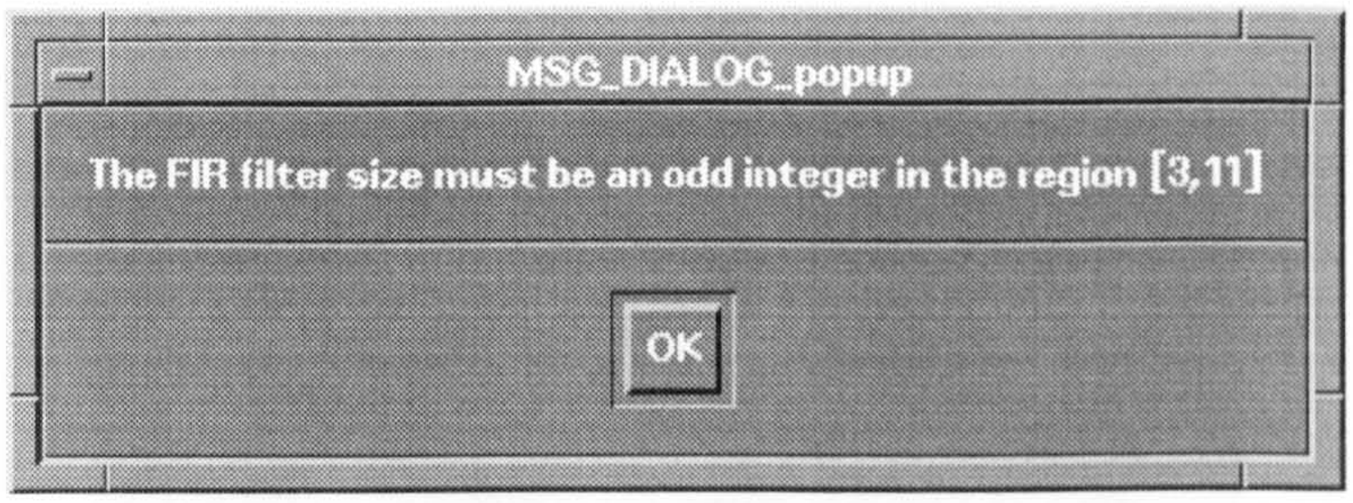
The user mistake



The DIP warning message



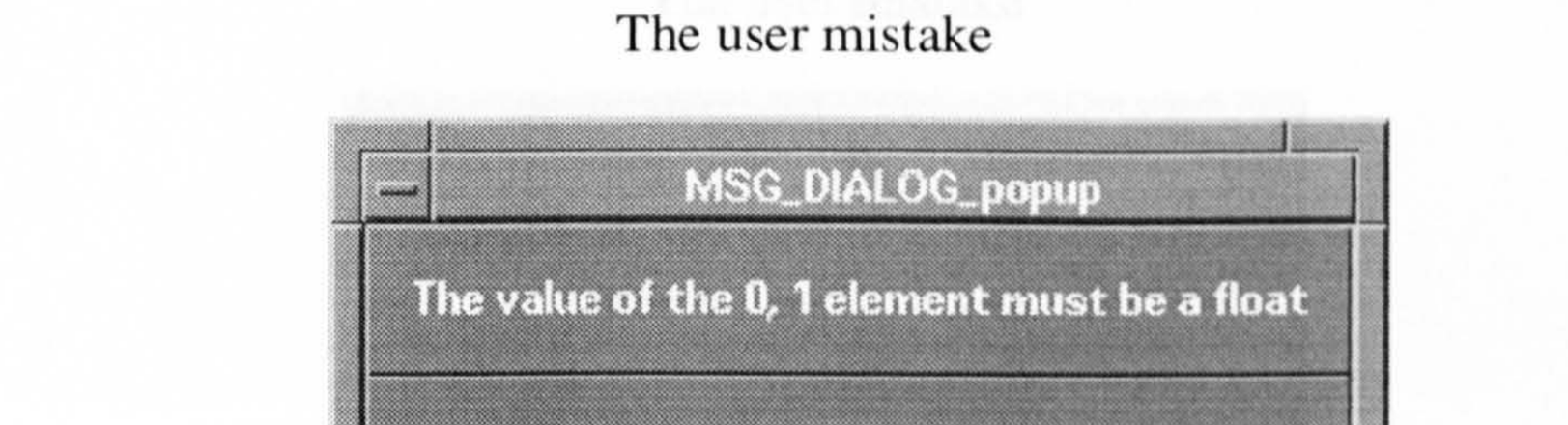
The user mistake



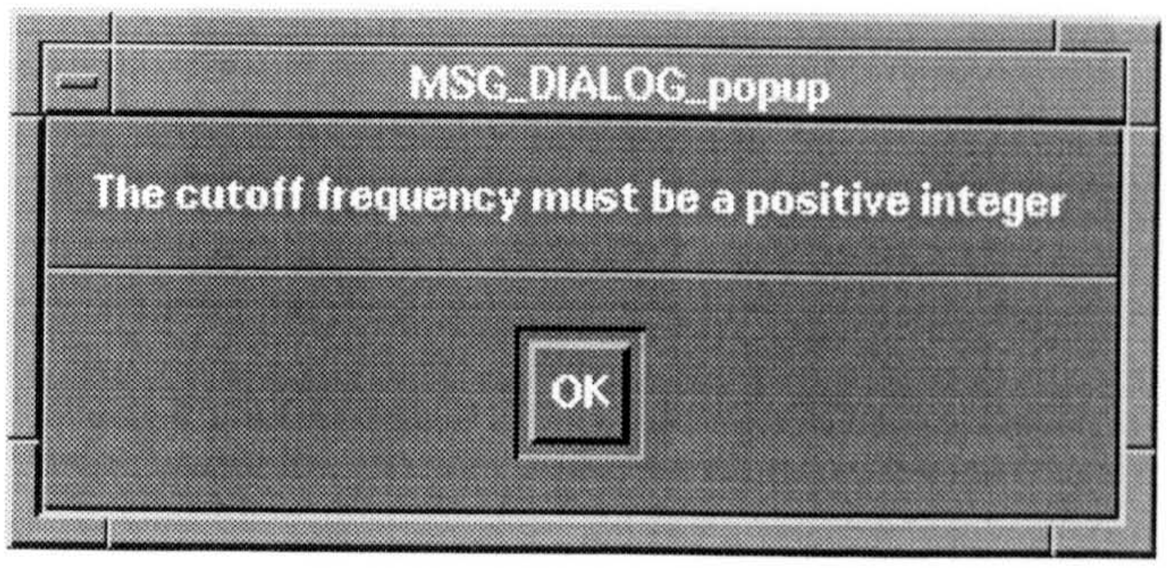
The DIP warning message

The user mistake

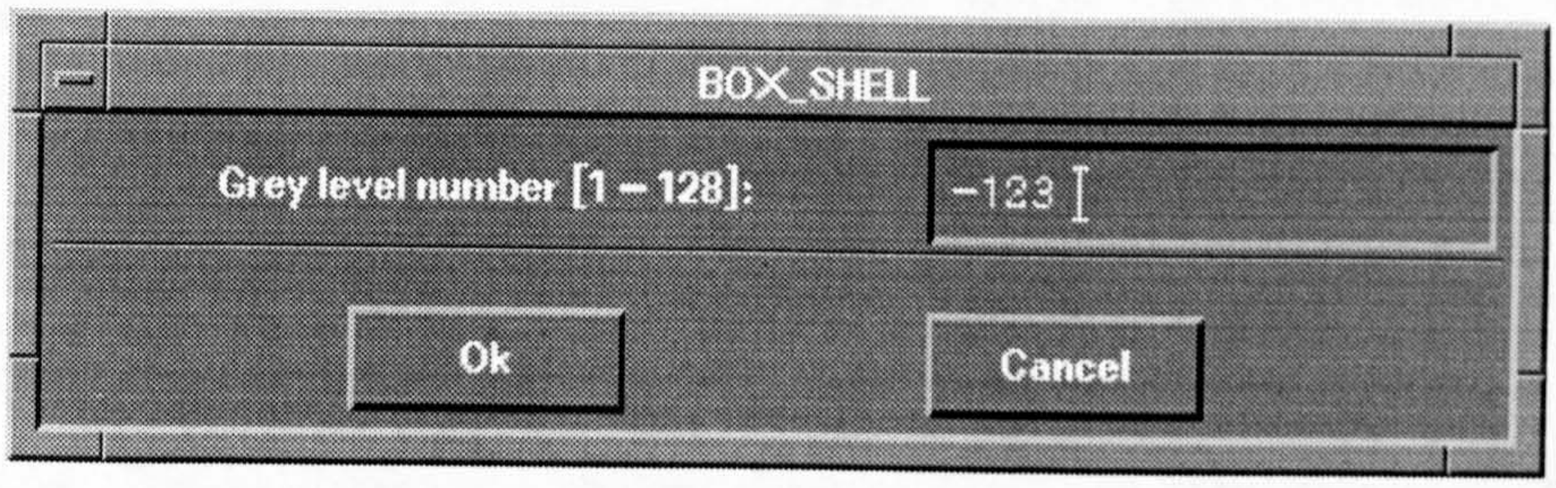
The DIP warning message



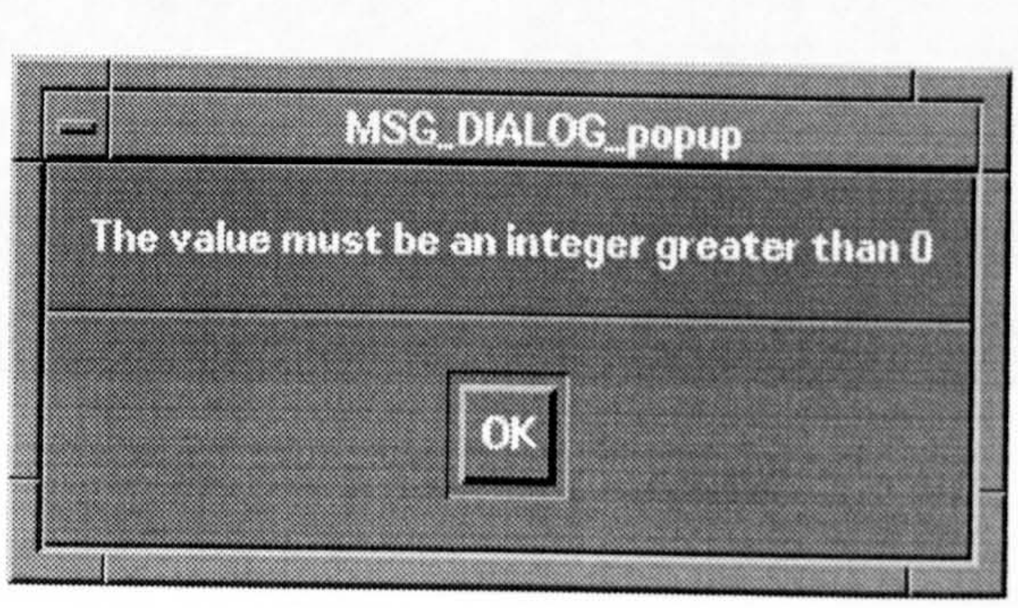
The user mistake



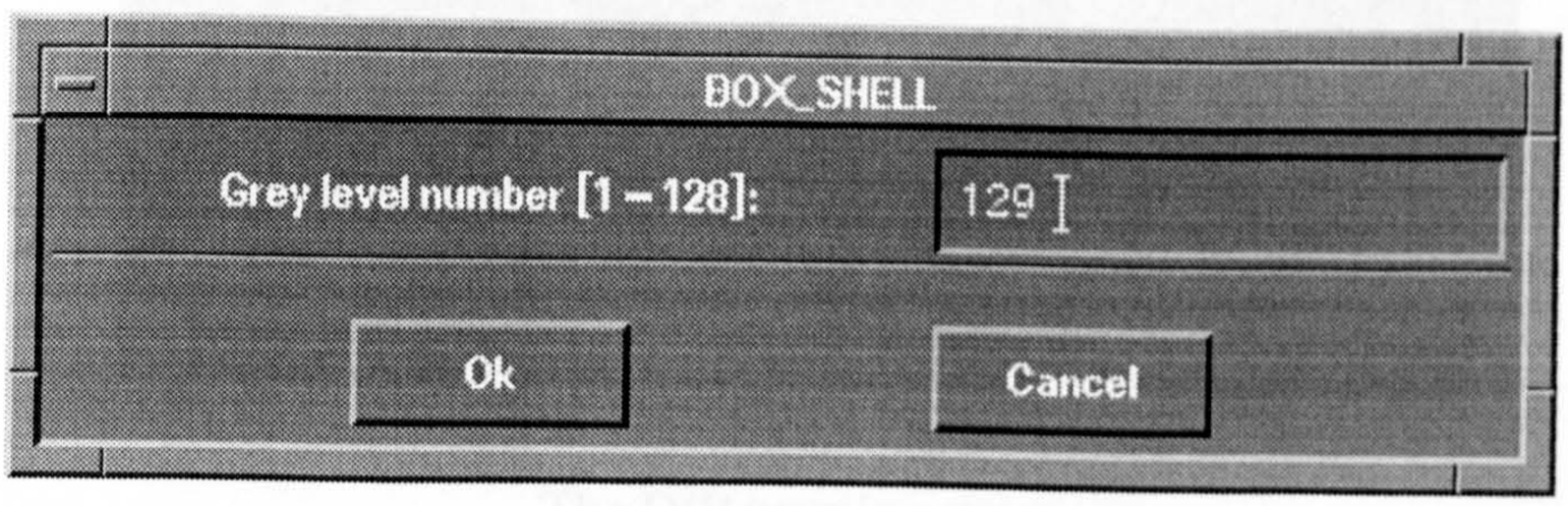
The DIP warning message



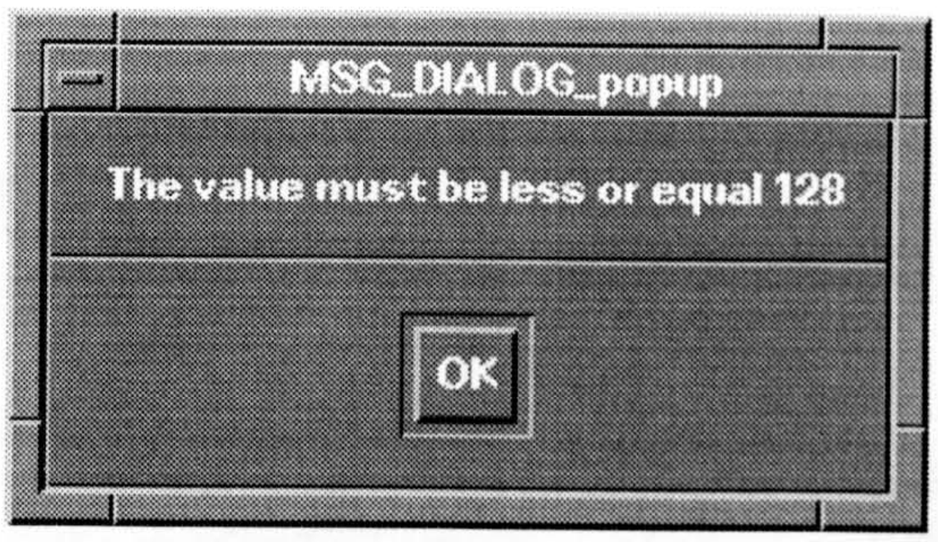
The user mistake



The DIP warning message

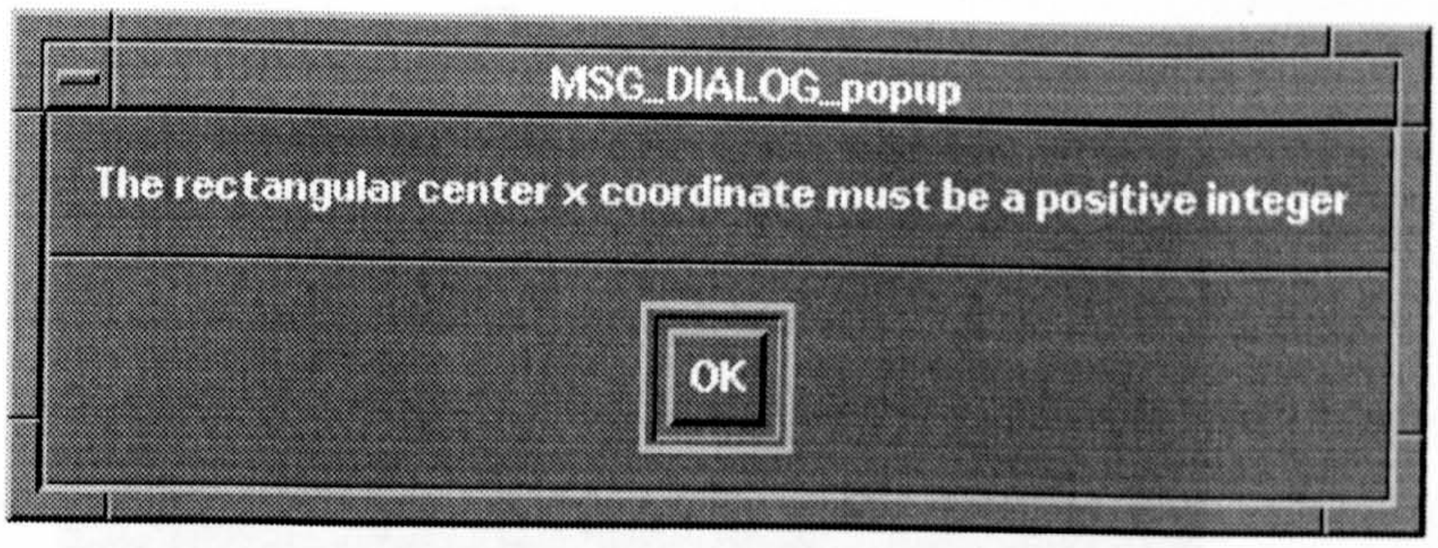


The user mistake

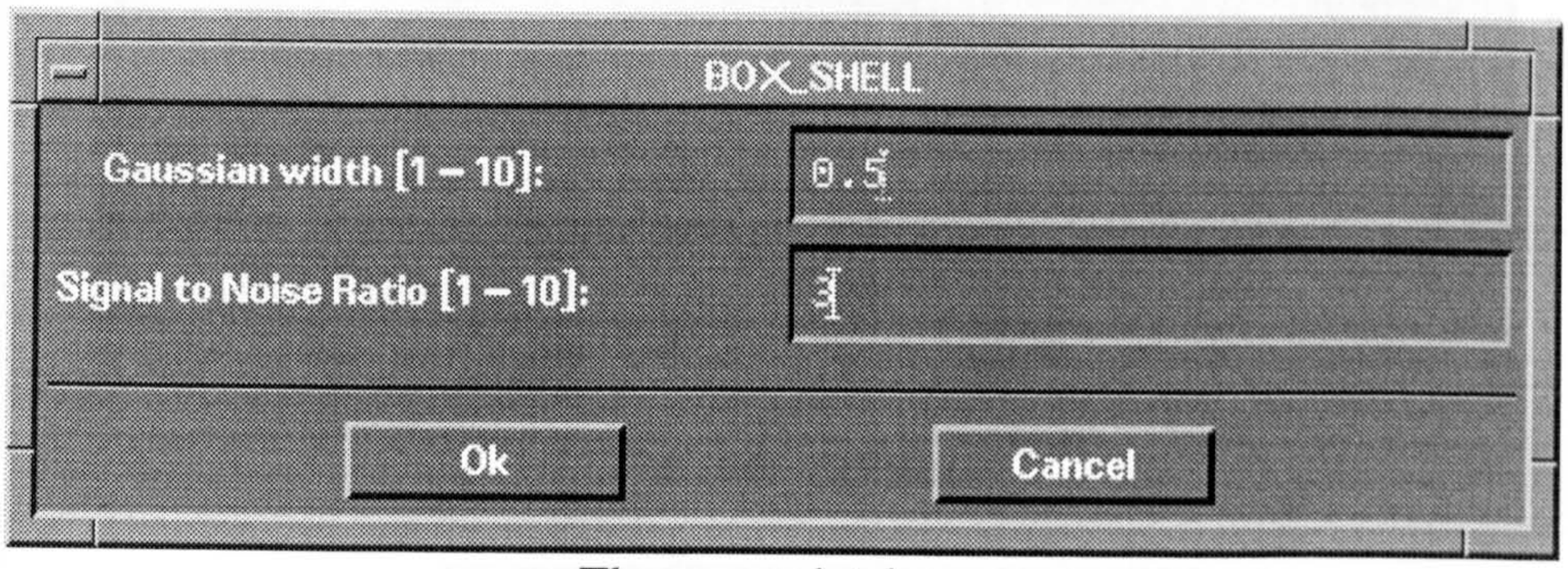


The DIP warning message

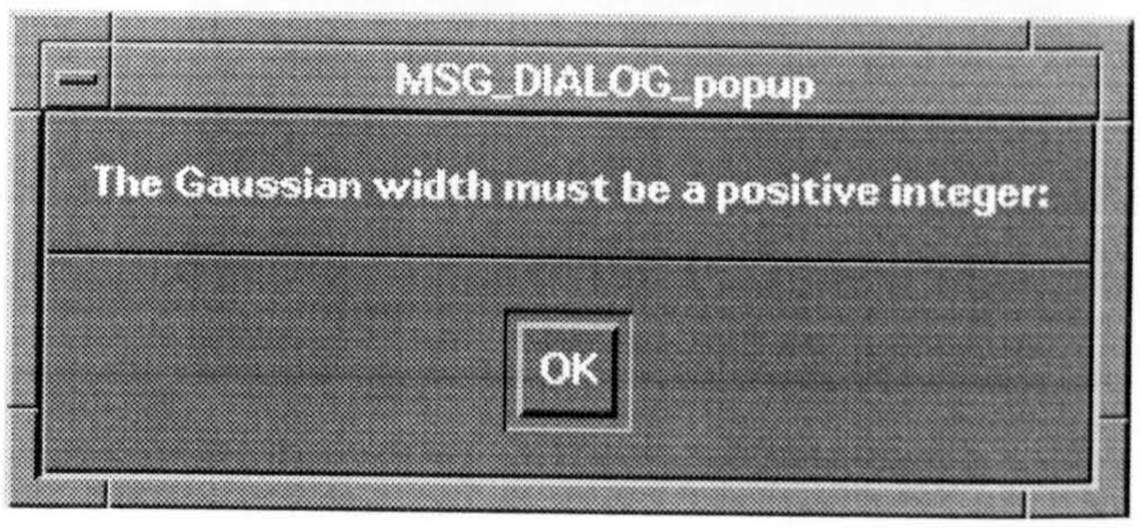
The user mistake



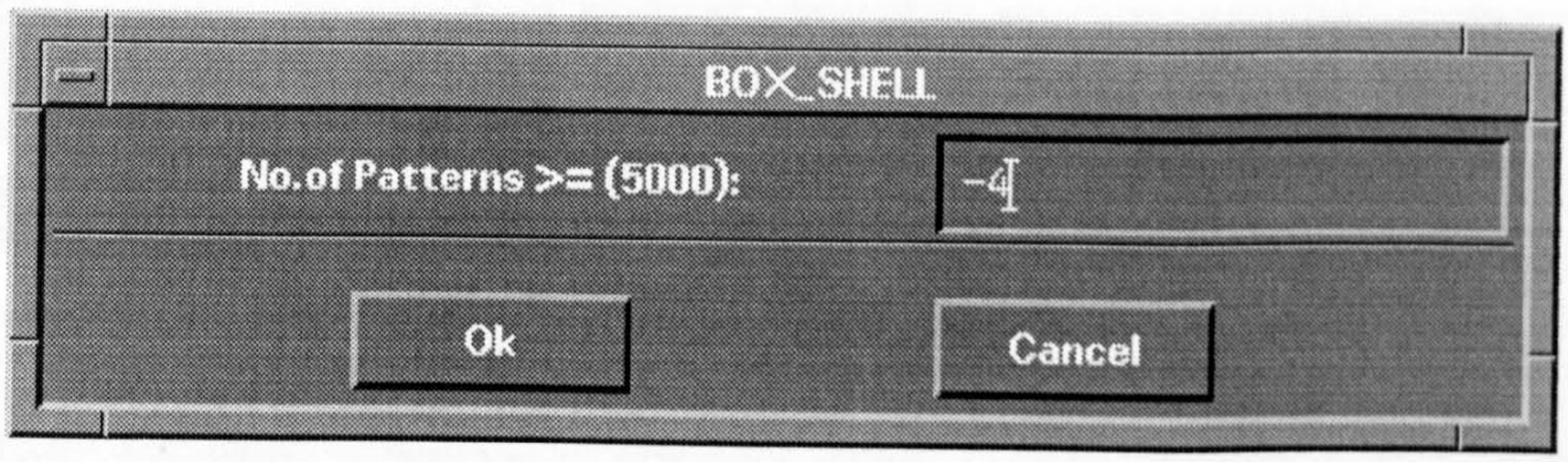
The DIP warning message



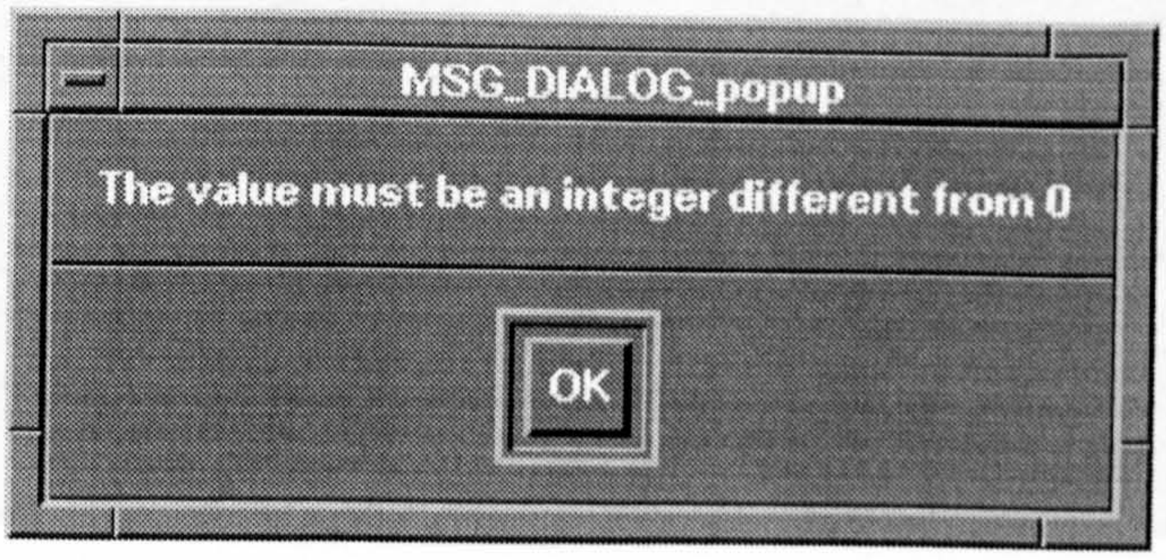
The user mistake



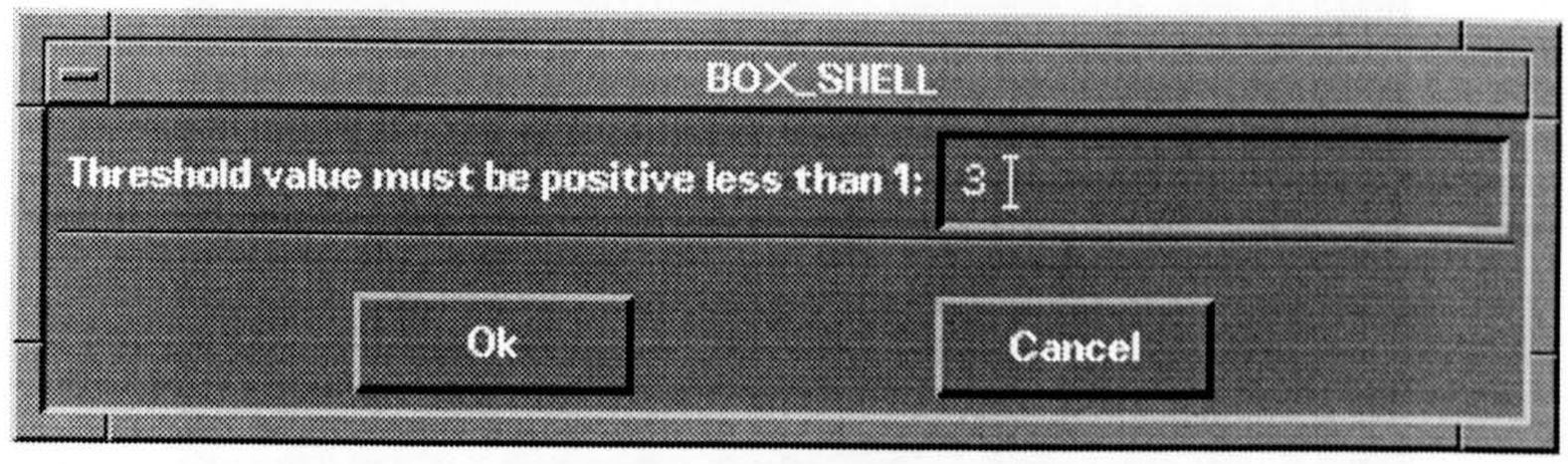
The DIP warning message



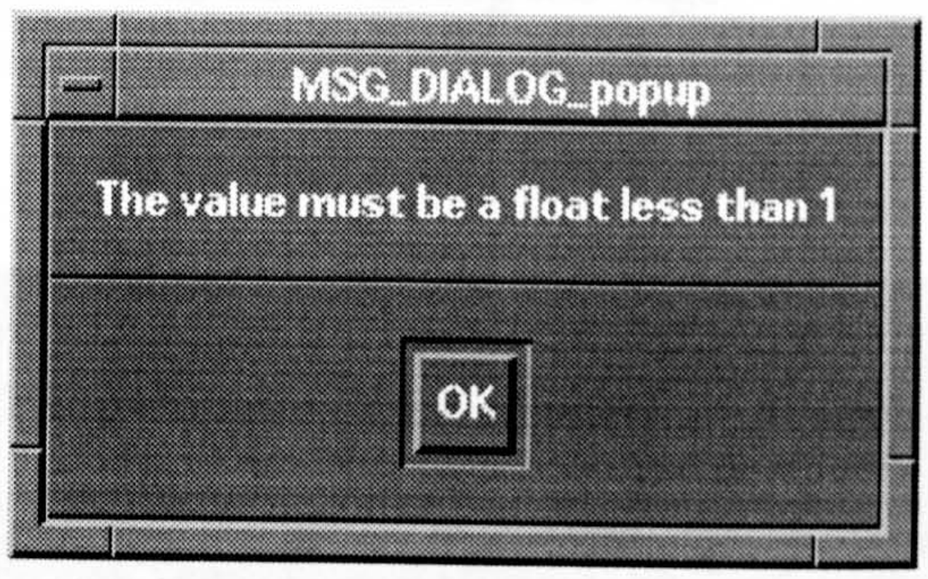
The user mistake



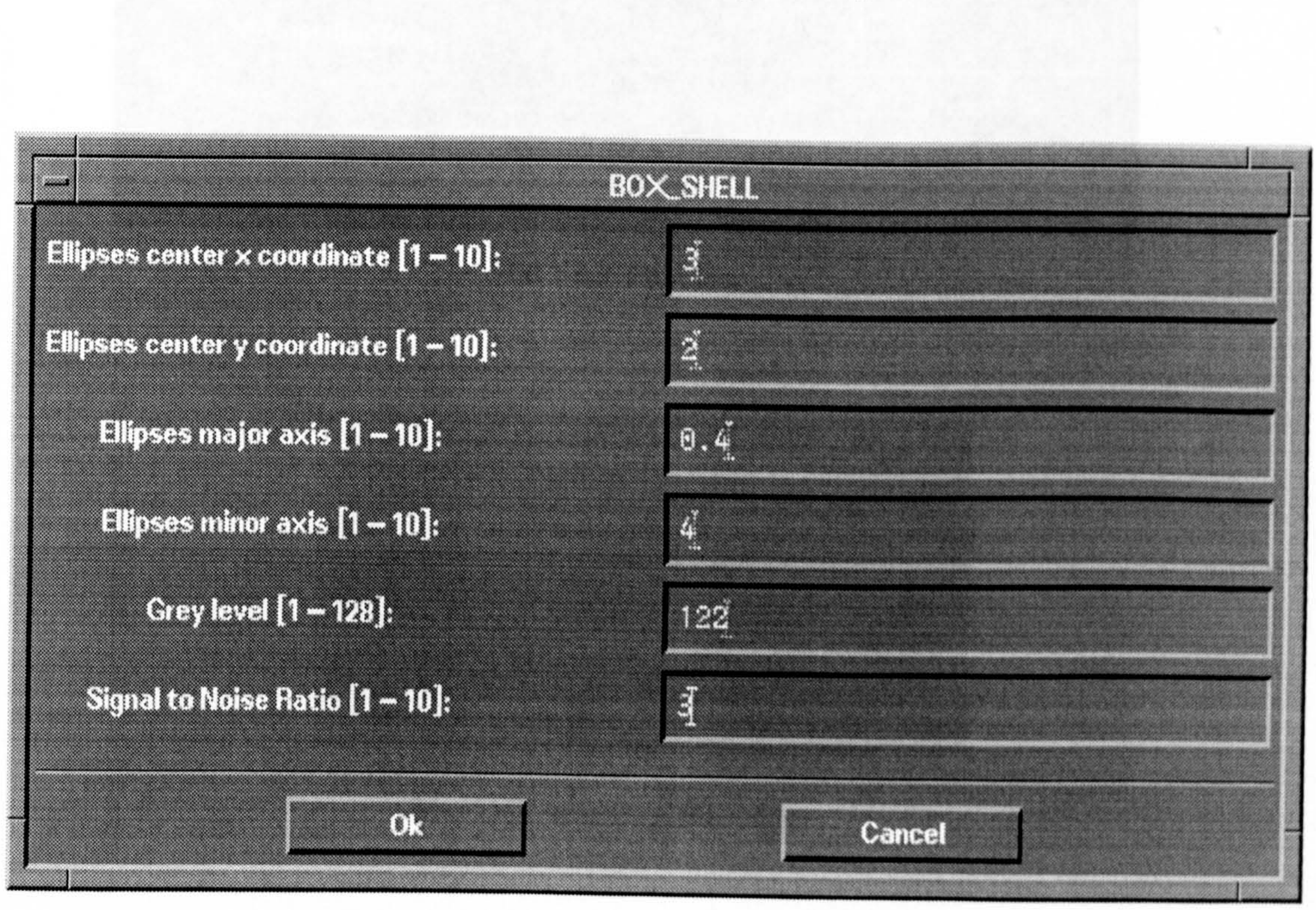
The DIP warning message



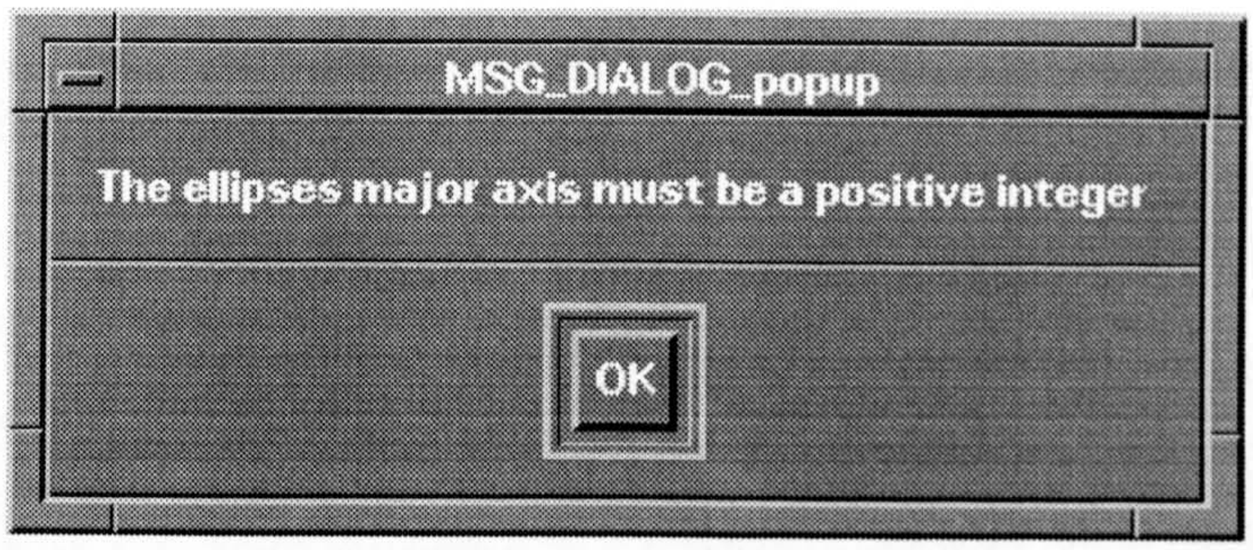
The user mistake



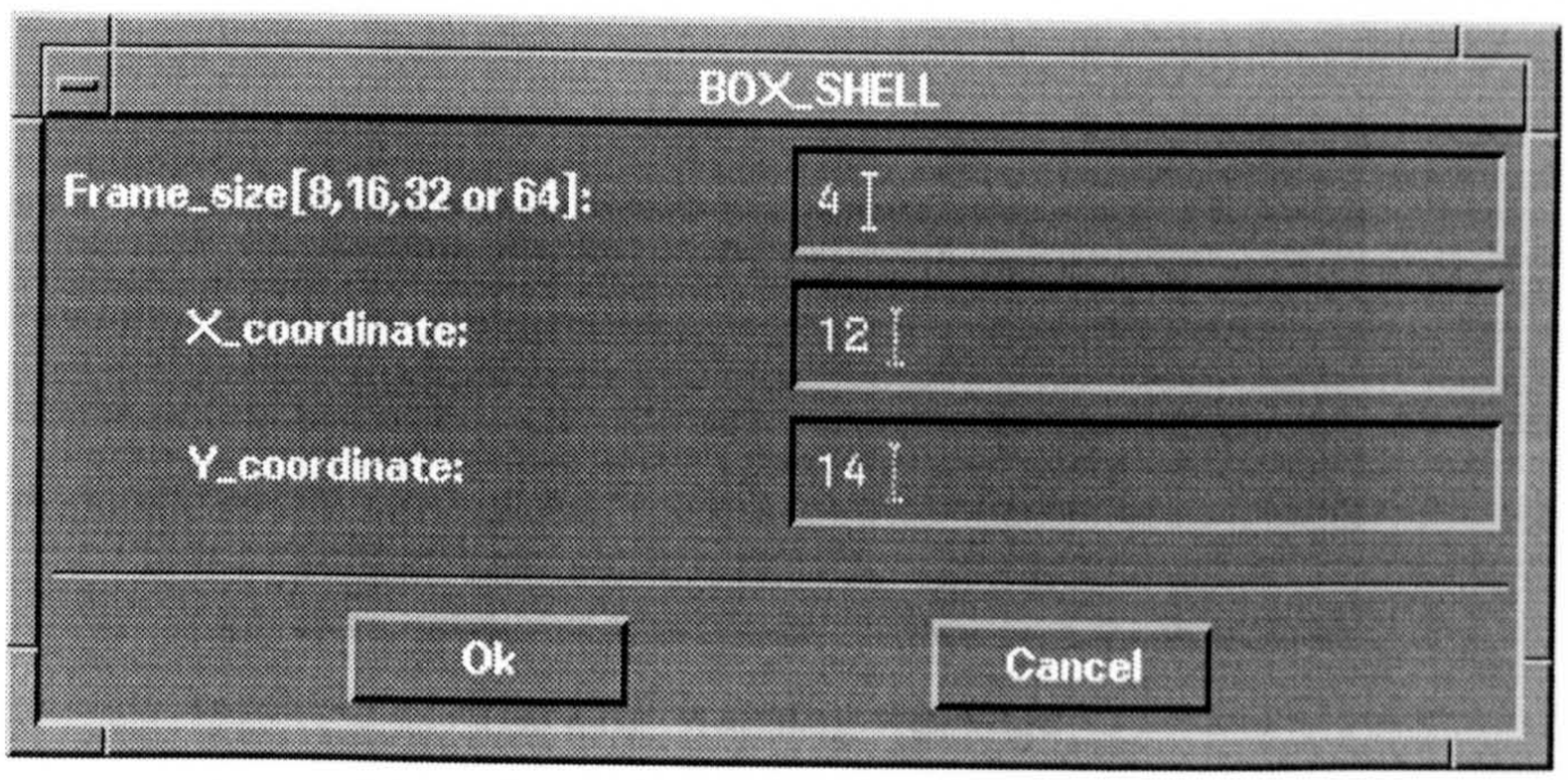
The DIP warning message



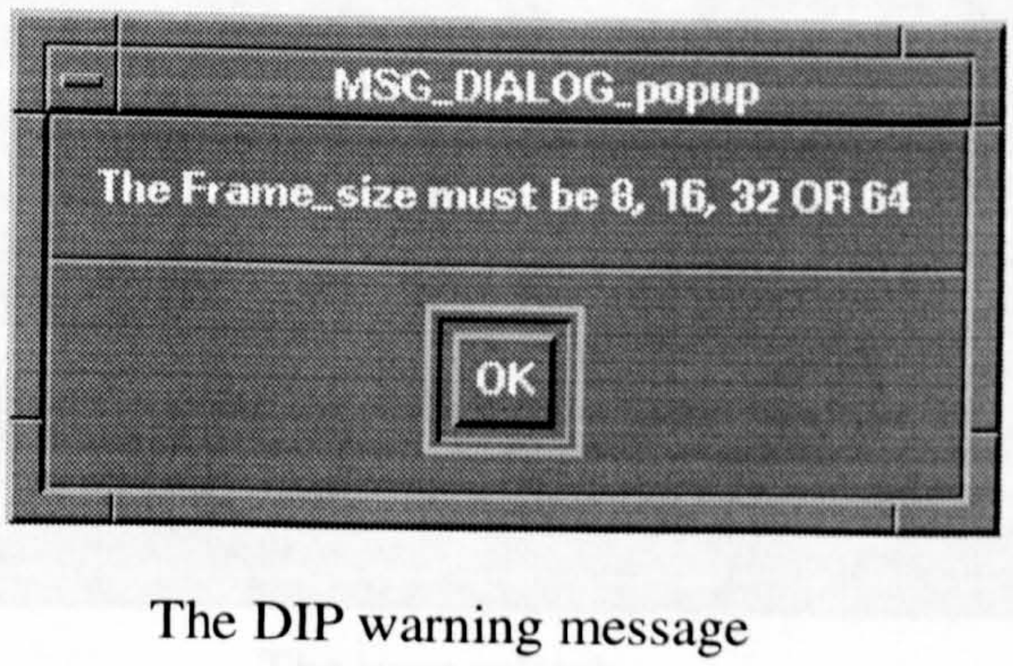
The user mistake



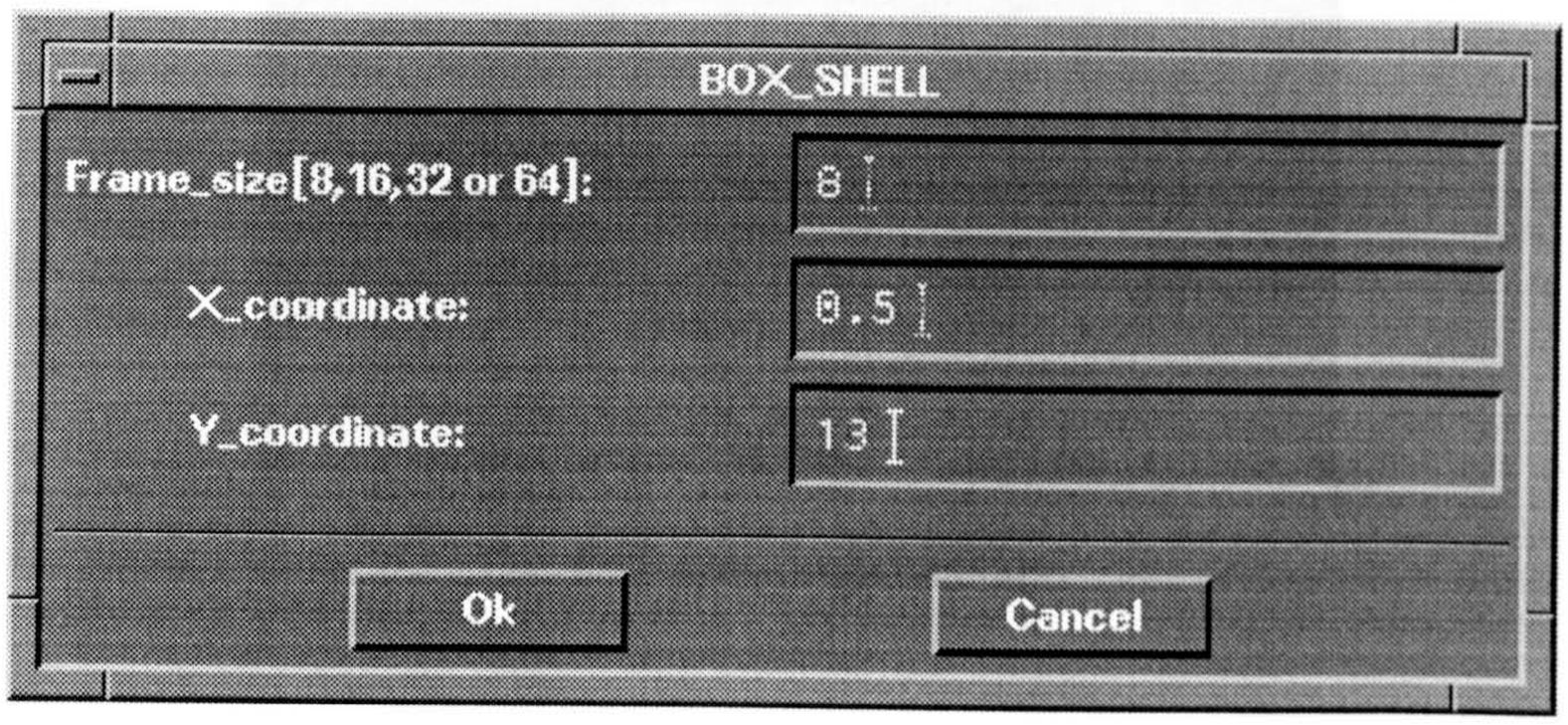
The DIP warning message



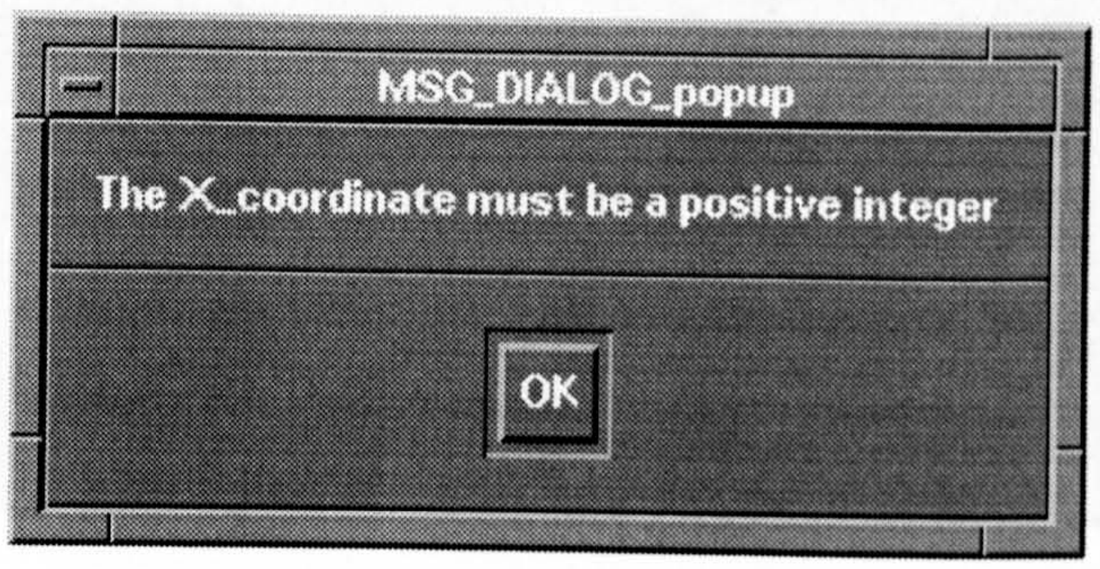
The user mistake



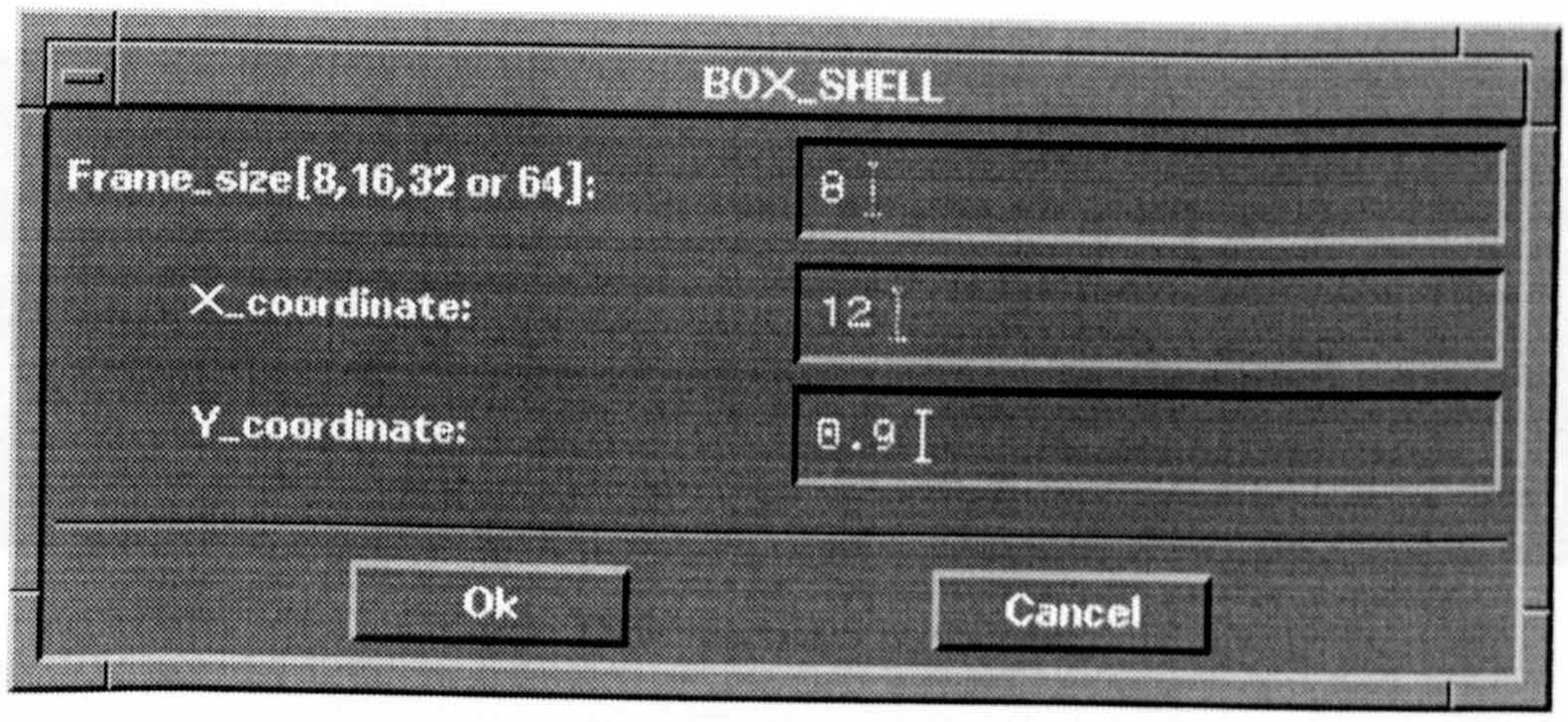
The DIP warning message



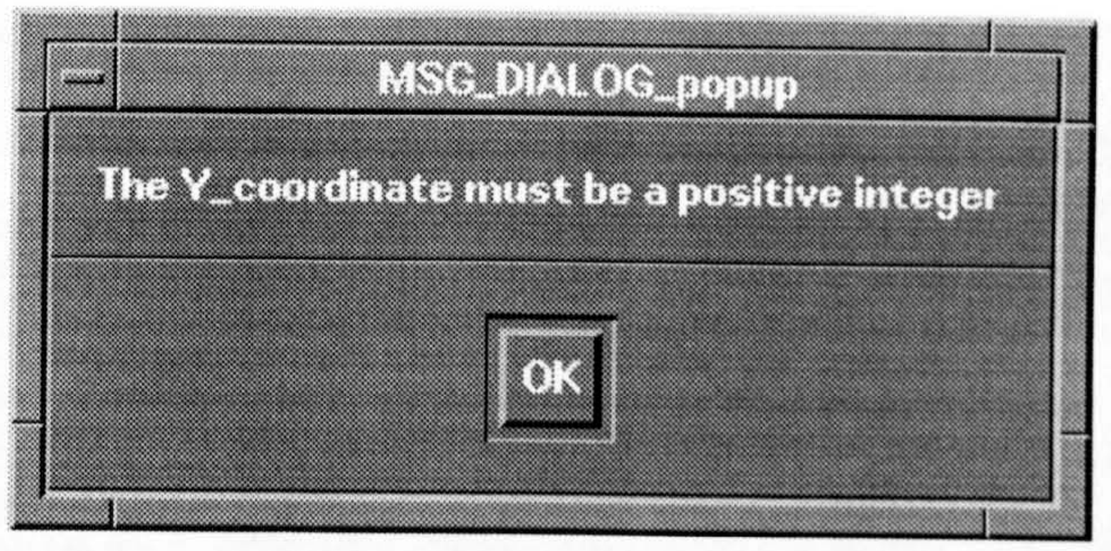
The user mistake



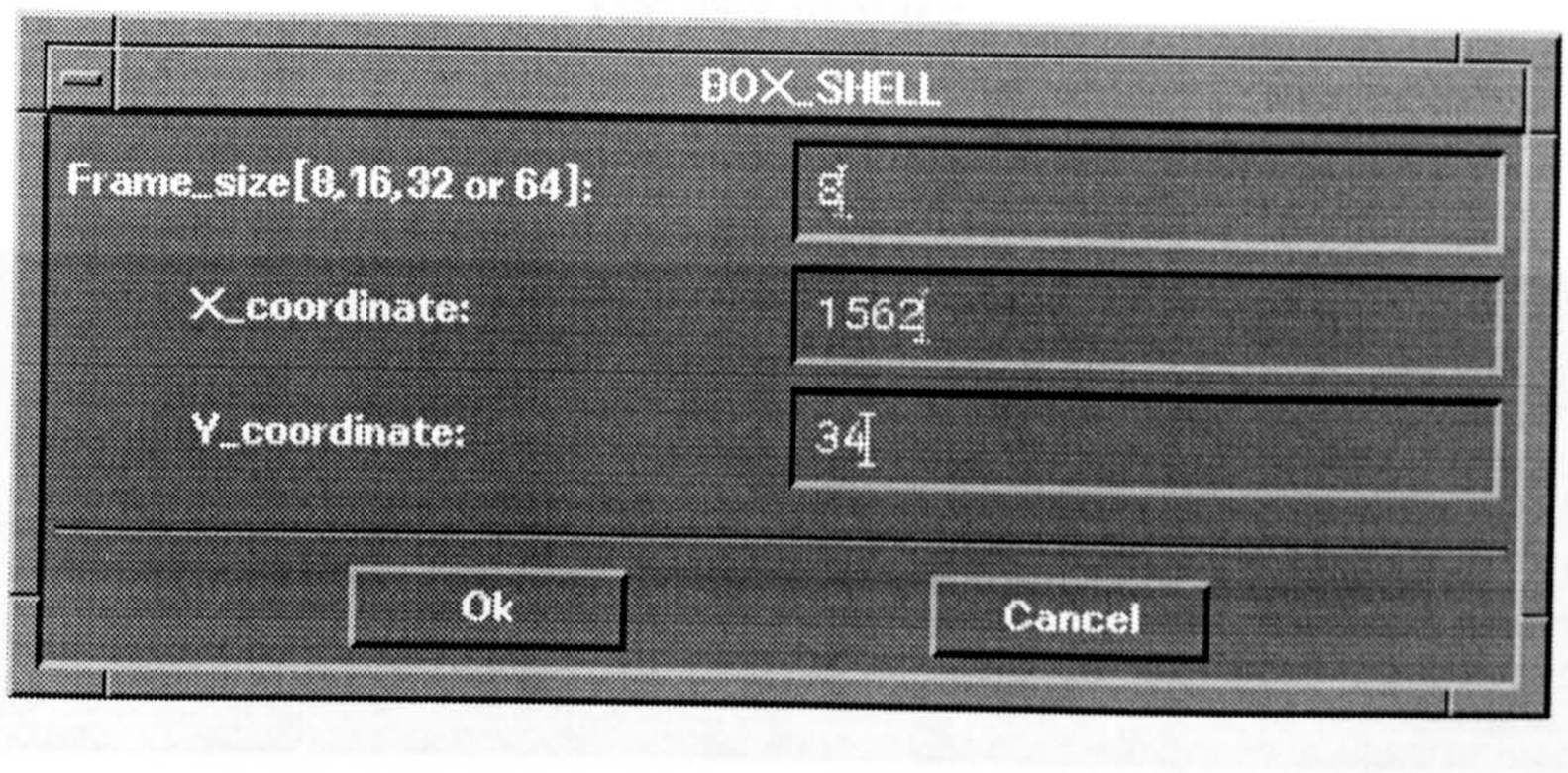
The DIP warning message



The user mistake



The DIP warning message



The user mistake

The DIP warning message

CONTAINS DISKETTE

UNABLE TO COPY

CONTACT UNIVERSITY

IF YOU WISH TO SEE

THIS MATERIAL